



PROCEEDINGS OF

THE AMERICAN MODELICA CONFERENCE 2018

October 9-10
Samberg Conference Center
Cambridge, MA
www.modelica.org



Proceedings of the 1st American Modelica Conference

Cambridge, Massachusetts, USA, October 9-10, 2018

Editors

Michael Tiller, Hubertus Tummescheit, and Luigi Vanfretti

Published by

Modelica Association and Linköping University Electronic Press

ISBN: 978-91-7685-148-7

Series: Linköping Electronic Conference Proceedings No. 154

ISSN: 1650-3686

eISSN: 1650-3740

DOI: 10.3384/ecp18154

Organized by

North America Modelica Users' Group

na.modelica-users.org

in co-operation with:

Modelica Association

c/o PELAB, Linköpings Univ.

SE-581 83 Linköping

Sweden

Conference location

Samberg Conference Center

Massachusetts Institute of Technology

Chang Building (E52)

50 Memorial Drive

Cambridge, MA 02139

USA

Copyright © Modelica Association, 2018



DR. JOHN F. MCKIBBEN

Technology Section Head
Procter and Gamble

BIO

John McKibben, PhD, is Section Head in Baby Care Modeling & Simulation for Procter & Gamble (P&G), a global consumer products innovator where, since 2014, he has led a team focused on driving adoption of existing M&S tools by product and process development team.

These efforts were instrumental in significant reduction in physical testing budgets. From 2007 to 2014 he led a multi-disciplinary M&S team within P&G's Family Care business unit where, under his leadership, the team became one of the benchmarks within P&G for integrating M&S into business processes.

His work as the founding leader of a multi-faceted M&S team in P&G's Fabric and Home Care Modeling & Simulation business (2001–2007) resulted in a 10x increase in business benefit and 5x increase in staffing. His leadership of P&G's Corporate Modeling & Simulation CFD team (1994–2001) led to team staffing growth based on demonstrated process and product improvements.

Among the M&S tools brought into play under John's leadership are computational fluid dynamics, finite element analysis, process modeling and reliability simulation. John's career leadership accomplishments were recognized in October 2016 by P&G with his induction into the PRISM Society, P&G's highest engineering mastery award.

Dr. McKibben earned the PhD from the Institute of Paper Science and Technology (Georgia Institute of Technology) with an emphasis on modeling aerodynamically driven instabilities at free surfaces. His master's degree, also from the Institute of Paper Science and Technology, involved research on convergence acceleration of sequential modular process simulations. He earned bachelor's degrees in Chemical Engineering from Oregon State University and in Chemistry from Southern Oregon University.

ABSTRACT

Simulation Led Innovation at Procter & Gamble

The required pace of innovation continues to increase in the consumer packaged goods industry. Increasingly historical approaches of development by physical prototypes lead to slow execution, high costs and incremental innovations.

By leveraging simulations we can deliver better innovations, faster, at lower costs. Succeeding on this journey requires more than just technical mastery. Changing the way that innovation is done also requires changing how decisions are made and how resources are allocated. It is also necessary to think about the entire lifecycle of the digital assets such as simulation platforms in a manner similar to how we manage our physical assets (e.g. pilot plants). Digital assets required continued funding, maintenance and upgrades to remain viable over time.



DR. HILDING ELMQVIST

CEO
Mogram AB

BIO

Dr. Hilding Elmqvist attained his Ph.D. at the Department of Automatic Control, Lund Institute of Technology, Sweden in 1978. His Ph.D. thesis contains the design of a novel object-oriented model language called Dymola and algorithms for symbolic model manipulation. It introduced a new modeling methodology based on connecting submodels according to the corresponding physical coupling instead of using signal flows. Submodels were described declaratively by equations instead of assignment statements.

Elmqvist spent one year in 1978-1979 at the Computer Science Department at Stanford University, California. His research continued in 1979-1984 on languages for implementation of control systems and for visual modeling. Elmqvist was in 1984-1990 the principal designer and project manager at a subsidiary to Alfa-Laval called SattControl in Malmö for developing SattGraph, a user interface system for process control and SattLine, a graphical, object-oriented and distributed control system. In 1990-1992, he worked for Alfa-Laval in Toronto.

In 1992, Elmqvist founded Dynasim AB in Lund, Sweden. Their primary product is Dymola for object-oriented modeling allowing graphical composition of models and 3D visualization of model dynamics.

Elmqvist took the initiative in 1996 to organize an international effort to design the next generation object-oriented language for physical modeling, Modelica. In April 2006, Dynasim AB was acquired by Dassault Systèmes. Elmqvist was the worldwide Chief Technology Officer for CATIA Systems within Dassault Systèmes until December 2015.

In January 2016, Elmqvist founded Mogram AB. Current activities include designing and implementing an experimental modeling language called Modia, based on the Julia language.

Hilding Elmqvist currently works part-time as a Technical Fellow at Modelon.

ABSTRACT

Modelica - History, State, Needs, Trends, and Possibilities

Model-based product design requires an intuitive and effective user interface, a standard language to encode models, high-fidelity model libraries for reuse, standardized format for simulation deployment, automated workflows and large computing power. The presentation will contain a brief history of Modelica evolution and current state including some applications. Some new needs will be discussed such as virtual testing of autonomous vehicles. New technical possibilities will be introduced, such as web apps for intuitive and effective user interaction and easy deployment and access, domain-specific language extensions for advanced modeling capabilities and cloud computing for large-scale simulation deployment.

KEYNOTE SPEAKER

PROGRAM COMMITTEE

Conference Chairs

Dr. Michael Tiller, Xogeny
Dr. Hubertus Tummescheit, Modelon

Program Chair

Prof. Luigi Vanfretti, Rensselaer Polytechnic Institute

Conference Board

Dr. Christopher Laughman, Mitsubishi Electric Research Laboratories
Dr. Michael Wetter, Lawrence Berkeley National Laboratory
Paul Goossens, Maplesoft
Behnam Afsharpoya, Dassault Systèmes

Program Committee

Dr. Johan Åkesson, Modelon AB, Lund, Sweden
Dr. Krystof Arendt, University of Southern Denmark, Denmark
Prof. Bernhard Bachmann, Univ. Applied Sciences Bielefeld, Bielefeld, Germany
Prof. John Baras, Univ. Maryland, USA
Dr. John Batteh, Modelon Inc., Ann Arbor, USA
Dr. Albert Benveniste, INRIA, Rennes, France
Christian Bertsch, Robert Bosch GmbH, Stuttgart, Germany
Volker Beuter, VI-grade GmbH, Marburg, Germany
Dr. David Blum, Lawrence Berkeley National Lab
Dr. Scott Bortoff, MERL Cambridge, USA
Dr. Timothy Bourke, INRIA, Paris, France
Daniel Bouskela, EDF R&D, Paris, France
Prof. David Broman, KTH, Stockholm, Sweden
Dr. Felix Buening, EMPA / ETH Zürich, Switzerland
Dr. Dan Burns, MERL, Cambridge, USA
Dr. Yan Chen, Pacific Northwest Laboratory
Prof. Francesco Casella, Politecnico di Milano, Milano, Italy
Prof. Massimo Cimmino McGill University, Montreal, Canada
Dr. Johan de Kleer, Xerox Parc, Palo Alto, USA
Mike Dempsey, Claytex Services Ltd, UK
Dr. Olaf Enge-Rosenblatt, Fraunhofer IIS, Dresden, Germany
Jens Frenkel, ESI-ITI GmbH, Dresden, Germany
Dr. Jens Frenkel, ESI ITI GmbH, Dresden, Germany
Dr. Yutaka Hirano, Toyota Motor Corporation, Japan
Dr. Jianjun Hu, Lawrence Berkeley National Laboratory, Berkeley, USA
Prof. Bengt Jacobson, Chalmers Technical University, Gothenburg, Sweden
Dr. Filip Jorissen, KU Leuven, Leuven, Belgium
Dr. Christian Kral, TGM, Vienna, Austria
Dr. Chris Laughman, MERL, Cambridge, USA
Prof. Alberto Leva, Politecnico di Milano, Milano, Italy
Dr. Alessandro Maccarini, Aalborg University, Aalborg, Denmark
Kristin Majetta, Fraunhofer IIS, Dresden, Germany
Dr. Alexandra Melhase, TU Berlin, Berlin, Germany
Dr. Lars Mikelsons, Bosch-Rexroth GmbH, Lohr am Main, Germany
Prof. Henrik Nilsson, University of Nottingham, Nottingham, Great Britain
Thierry Stéphane Noudui, Lawrence Berkley National Lab, Berkeley, USA

Dr. Hans Olsson, Dassault Systèmes, Lund, Sweden
Zheng O'Neill, The University of Alabama, Alabama, USA
Prof. Martin Otter, DLR, Oberpfaffenhofen, Germany
Kaustubh Phalak, Ingersoll Rand Corporation, Minneapolis, USA
Johan Rhodin, 84 Codes Consulting LLC, Missouri, USA
Lisa Rivalin, Engie Axima, Paris, France
Dr. Clemens Schlegel, Schlegel Simulation, Munich, Germany
Prof. Gerhard Schmitz, Technical University Hamburg-Harburg, Germany
Dr. Peter Schneider, Fraunhofer IIS EAS, Dresden, Germany
Mathieu Schuman, EDF, Paris, France
Dr. Michael Sielemann, Modelon, Munich, Germany
Dr. Ed Tate, Exa, Livonia, USA
Dr. Wilhelm Tegethoff, TLK-Thermo GmbH and TU Braunschweig, Germany
Dr. Matthias Thorade, Berlin University of the Arts, Berlin, Germany
Dr. Michael Tiller, Xogeny, Michigan, USA
Dr. Jakub Tobolar, DLR, Oberpfaffenhofen, Germany
Dr. Hubertus Tummescheit, Modelon Inc, West Hartford, USA
Prof. Alfonso Urquía, UNED, Madrid, Spain
Dr. Gavan Valentin, Engie Lab, Paris, France
Prof. Luigi Vanfretti, Rensselaer Polytechnical Institute, Troy, NY, USA
Prof. Hans Vangheluwe, McGill University, Canada and University of Antwerp, Belgium
Dr. Subbarao Varigonda, Cummins, Columbus, USA
Dr. Michael Wetter, Lawrence Berkeley National Laboratory, Berkeley, USA
Dr. Stefan Wischhusen, XRG, Hamburg, Germany
Prof. Dietmar Winkler, University College of Southeast Norway, Norway
Prof. Wangda Zuo, University of Miami, Miami, USA

CONTENTS

Session 1 / Room 1 THERMOFLUIDS 1

| | |
|---|----|
| Investigation of Fuel Reduction Potential of a Capacity Controlled HVAC System for Buses Using Virtual Test Drives | 7 |
| Christian Kaiser, Sebastian Meise, Wilhelm Tegethoff, Jürgen Köhler | |
| Control Description Language | 17 |
| Michael Wetter, Milica Grahovac, Jianjun Hu | |
| Molten Salt-Fueled Nuclear Reactor Model for Licensing and Safeguards Investigations | 27 |
| Scott Greenwood | |

Session 1 / Room 2 AEROSPACE

| | |
|---|----|
| Development and Implementation of a Flexible Model Architecture for Hybrid-Electric Aircraft | 37 |
| John Batteh, Jesse Gohl, Michael Sielemann, Peter Sundstrom, Ivar Torstensson, Natesa MacRae, Patrick Zdunich | |
| A Modelica Library for Spacecraft Thermal Analysis | 46 |
| Tobias Posielek | |
| Exergy Analysis of Thermofluid Energy Conversion Systems in Model-Based Design Environment | 56 |
| Daniel Bender | |

Session 2 / Room 1 THERMOFLUIDS 2

| | |
|--|----|
| On Closure Relations for Dynamic Vapor Compression Cycle Models | 67 |
| Christopher Laughman and Hongtao Qiao | |
| Fast Calculation of Refrigerant Properties in Vapor Compression Cycles Using the Spline-Based Table Look-Up Method (SBTL) | 77 |
| Lixiang Li, Jesse Gohl, John Batteh, Christopher Greiner, Kai Wang | |
| Modelica-Based Dynamic Modeling of a Solar Powered Ground Source Heat Pump System | 85 |
| Defeng Qian, Zheng O'Neill | |

Session 2 / Room 2 ENERGY SYSTEMS

| | |
|--|-----|
| Coalesced Gas Turbine and Power System Modeling and Simulation using Modelica | 93 |
| Miguel Aguilera, Luigi Vanfretti, Francisco Gómez, Tetiana Bogodorova | |
| Analysing the Stability of an Islanded Hydro-Electric Power System | 103 |
| Dietmar Winkler | |
| Modeling of PMU-Based Islanded Operation Controls for Power Distribution Networks using Modelica and OpenIPSL | 112 |
| Biswarup Mukherjee, Luigi Vanfretti | |

Session 3 / Room 1 TOOLS & FMI

| | |
|---|------------|
| ModestPy: An Open-Source Python Tool for Parameter Estimation in Functional Mock-up Units | 121 |
| Krzysztof Arendt, Muhyiddine Jradi, Michael Wetter, Christian Veje | |
| A Safe Regression Test Selection Technique for Modelica..... | 131 |
| Niklas Fors, Jon Sten, Markus Olsson, Filip Stenström | |
| Functional Mockup Interface: An Empirical Survey Identifies Research Challenges and Current Barriers | 138 |
| Gerald Schweiger, Cláudio Gomes, Georg Engel, Irene Hafner, Thierry Noudui, Josef-Peter Schögl | |
| A Method to Import FMU to Hardware Description Language | 147 |
| Min Zhang | |

Session 3 / Room 2 MECHANICAL SYSTEMS

| | |
|--|------------|
| Developing a Framework for Modeling Underwater Vehicles in Modelica..... | 157 |
| Shashank Swaminathan, Srikanth Saripalli | |
| Hybridisation and Splitting of a Crank Angle Resolved Internal Combustion Engine Model Using a Mean Value Intake for Real-Time Performance..... | 165 |
| Xiaoran Han, Alessandro Picarelli, Mike Dempsey, Romain Gillot | |
| Component-Based 3D Modeling Combined with Equation-Based Modeling..... | 175 |
| Andrea Neumayr, Martin Otter | |
| The Deployable Structures Library..... | 187 |
| Cory Rupp, Laura Schweizer | |

Session 5 / Room 1 TOOLS 2

| | |
|--|------------|
| Modelica Language - A Promising Tool for Publishing and Sharing Biomedical Models..... | 196 |
| Jiří Kofránek, Filip Ježek, Marek Mateják | |
| The OpenModelica Integrated Modeling, Simulation and Optimization Environment..... | 206 |
| Peter Fritzsón, Adrian Pop | |
| Modelica on the Web..... | 220 |
| Tamas Kecskes, Patrik Meijer, Janos Sztipanovits, Peter Fritzsón, Adrian Pop, Arunkumar Palanisamy | |

Session 5 / Room 2 LIBRARIES

| | |
|--|------------|
| A Modelica Library for Thin-Layer Drying of Agricultural Products | 227 |
| Augusto Souza, Brian Steward, Carl Bern | |
| Modelica Library for the Systems Engineering of Railway Brakes | 236 |
| Marc Ehret | |
| Drilling Library: A Modelica Library for the Simulation of Well Construction | 246 |
| Reza Dadfar, Stéphane Velut, Per-Ola Larsson, Mathias Strandberg, Håkan Runvik, Johan Windahl, Pål Kittilsen, John-Morten Godhavn, Åsmund Hjulstad | |

Investigation of fuel reduction potential of a capacity controlled HVAC system for buses using virtual test drives

Christian Kaiser¹ Sebastian Meise² Wilhelm Tegethoff¹ Jürgen Köhler²

¹TLK-Thermo GmbH, Germany, {c.kaiser, w.tegethoff}@tlk-thermo.com

²Institut für Thermodynamik, TU-Braunschweig, Germany, {s.meise, juergen.koehler}@tu-braunschweig.de

Abstract

The refrigerant cycle in conventional omnibus HVAC systems has a significant influence on fuel consumption and, as a result, on vehicle emissions. The additional emissions resulting from the use of the air conditioning system are called indirect emissions. In addition, there are so-called direct emissions from the air conditioning system caused by unintended leakage of refrigerant. A reduction in indirect emissions can be achieved, for instance, by adjusting the capacity of the refrigerant compressor. A reduction in direct emissions can be achieved by so-called alternative or natural refrigerants. To investigate approaches to reducing direct and indirect emissions, a total vehicle simulation model of a coach with detailed HVAC systems was developed with full implementation in Modelica. For this total vehicle simulation of a coach with a detailed HVAC system, a refrigerant cycle based on the natural refrigerant CO₂ (R-744) was modeled and validated. In addition, an efficient control strategy was developed by adjusting the capacity of the refrigerant compressor to cover the actual cooling capacity demand and save fuel. Based on virtual driving test scenarios, the fuel saving potential of the developed compressor capacity control strategy is investigated to determine average annual fuel savings.

Keywords: HVAC, MAC, Energy efficiency, Omnibus, Total vehicle simulation, Virtual test drive, R-744, Fuel saving, Compressor capacity control, Cooling capacity control, Thermal systems

1 Introduction

Reciprocating compressors with constant displacement are typically used in air conditioning systems for conventional buses. The compressor is usually driven directly by the internal combustion engine through the use of a belt drive and a magnetic clutch. Due to the speed-synchronous mechanical linkage to the engine and the constant compressor displacement, different control techniques and methods are needed to realize variable cooling capacity based on compressor capacity control for efficient use of the air conditioning system. For this purpose, a combination of speed control by means of a two-speed pulley gearbox based on a planetary gearbox and cylinder bank shutdown by suction gas interlock are developed. In addition, there are other techniques and methods to adapt the refrigerant compressor capacity for cooling capacity control of reciprocating compressors with constant displacement (see Kaiser, 2018): for instance, refrigerant compressor capacity control by cycling clutch operation with a magnetic clutch or compressor speed control by continuously variable transmission or by separate drive with an electric machine as well as the use of semi-hermetic electric compressors. The cycling clutch operation of the magnetic clutch causes uncomfortable fluctuations in the interior temperature. The compressor speed control by continuously variable transmission is complex and requires lot of installation space. The use of a separate electric machine or semi-

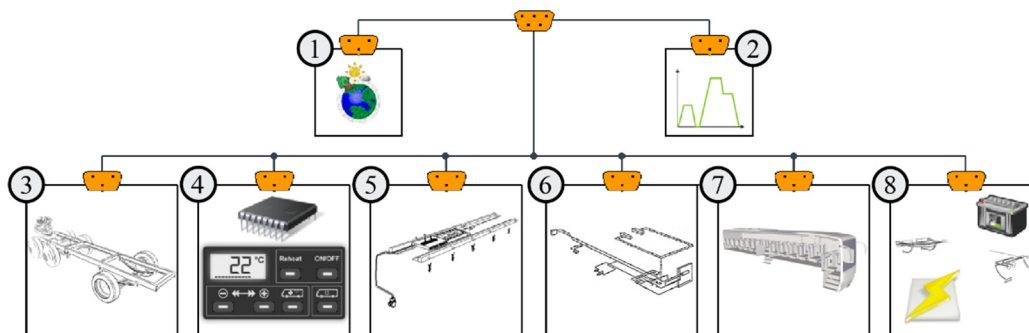


Figure 1. Modelica-based total vehicle simulation model of a coach: (1) Ambient conditions, (2) Driving conditions, (3) Vehicle longitudinal dynamics, (4) HVAC controller, (5) Refrigerant cycle, (6) Engine cooling and interior heating cycle, (7) Vehicle cabin and (8) Electrical system.

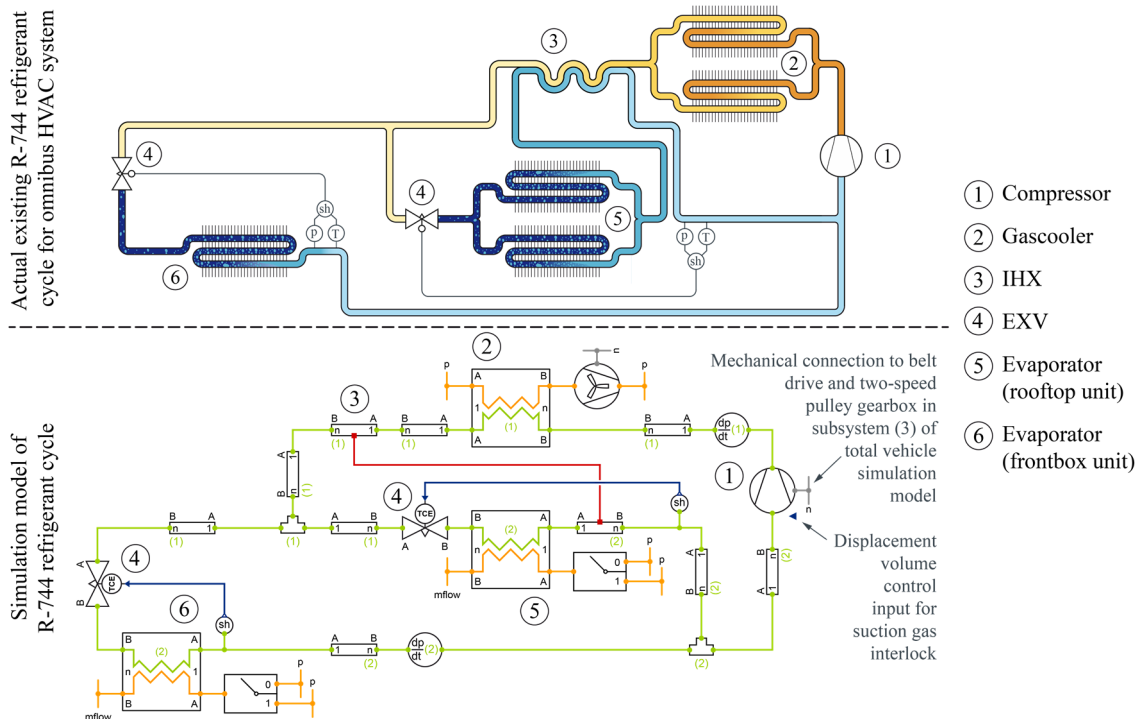


Figure 2. Structure of actual existing R-744 refrigerant cycle for omnibus HVAC system and topology of the corresponding R-744 refrigerant cycle simulation model based on (TIL Suite, 2018)

hermetic electric compressor requires lot of installation space as well. Generally, installation of electric driven compressors is more expensive than open-type mechanical compressors because of the additional need for more or more powerful electric generators and for a frequency converter for compressor speed adaption. Compared to a conventional mechanically driven compressor, the conversion from mechanical to electrical energy and back to mechanical energy for electric driven compressors leads to additional energy losses. Nevertheless, omnibus HVAC systems based on electrically driven compressors are currently being investigated, see (Meise et al, 2018; Hebeler et al, 2018). In contrast to the above-mentioned compressor capacity control techniques and methods, the combination of two-speed pulley gearbox and cylinder bank shutdown is not complex, requires nearly no further installation space and entails no additional energy losses through energy conversion. For this reason, the combination of two-speed pulley gearbox and cylinder bank shutdown is the best solution for use in conventional buses.

The study of this combination of two-speed pulley gearbox and cylinder bank shutdown is done by means of a detailed vehicle simulation of a coach. The vehicle model of a coach was developed and validated for research issues in the realm of air conditioning systems in buses, see (Kaiser, 2018). Figure 1 shows the overall model, which includes the following subsystems: ambient and driving conditions as boundary conditions, longitudinal driving dynamics, climate controller, refrigeration cycle, engine cooling and heating cycle, interior of the bus as well as the electrical system.

Special emphasis was put on detailed models with modeling of all fundamentally relevant heat transfers and pressure losses for two air conditioning systems based on refrigerant R-134a and R-744. A detailed description of modeling the R-744 refrigerant cycle appears below. Further on, the two-speed pulley gearbox based on a planetary gearbox and the cylinder bank shutdown by suction gas interlock are described briefly. After that, two climatically different driving route scenarios for the comparative study are introduced. To conclude, the numerical simulation results of the specified compressor capacity control method are presented depending on two driving route scenarios (Germany, Portugal/Spain).

2 Modeling of R-744 refrigerant cycle for coach HVAC system

The following section presents the structure of an actual existing R-744 refrigerant cycle for an omnibus HVAC system as well as the topology of the corresponding simulation model. Subsequently, the modeling of the R-744 refrigerant cycle based on selected heat transfer and pressure drop correlations is described and the results of the calibration and validation process are shown.

2.1 Modeling of R-744 refrigerant cycle

Figure 2 shows the structure of an actual existing R-744 refrigerant cycle for an omnibus HVAC system and the corresponding simulation model of the R-744 refrigerant cycle. The side-by-side illustration of the refrigerant cycle structures shows that the parallel heat

exchangers (gascooler and evaporator) of the rooftop unit are merged in one corresponding heat exchanger model in the R-744 refrigerant cycle model. In this merging process, the geometric, all thermal and hydraulic characteristics of the parallel heat exchangers are accounted for in the corresponding single heat exchanger models. For modeling of the R-744 refrigerant cycle, off-the-shelf models from the Modelica library (TIL Suite, 2018) were used (Richter, 2008; Gräber et al, 2010; Tegethoff et al, 2011, Schulze, 2013). The heat exchanger models of the gascooler and evaporator are modeled according to the finite volume method with 10 discrete volumes each. The IHX (internal heat exchanger) is also modeled according to the finite volume method, but with 4 discrete volumes. All of them used models from the Modelica library (TIL Suite, 2018) with a stream operator and all of them allow for reversal of flow and zero mass flow, see (Schulze, 2013). The models used are very robust and well suited for high dynamic simulation applications with VLE (vapor-liquid equilibrium) fluids, e.g. HVAC systems. The R-744 refrigerant cycle model shown in Figure 2 has 391 continuous time states and 19740 time-varying variables. The translated model of the R-744 refrigerant cycle has 2 linear equations, where the linear equation system size is 2 and 3. In addition, the translated model has 20 nonlinear equations, where each nonlinear equation system has size 1. Model translation and manipulation was done using Dymola.

The following describes the selected correlations for calculating thermal and hydraulic characteristics of the R-744 refrigerant cycle. Basically, all thermal and hydraulic calculations in all component models in the R-744 simulation model are based on geometric parameters of the existing R-744 refrigerant cycle. For the air side of all heat exchangers, the calculation of convective heat transfer and pressure drop is based on the calculation published by (Haaf, 1988). On the refrigerant side of the heat exchangers, the refrigerant undergoes phase change processes. For this purpose, specific correlations are implemented. For the single-phase refrigerant in laminar flow region ($Re < 2300$), the heat transfer coefficient is determined using the constant Nusselt number of $Nu = 3,657$. For the single-phase

refrigerant in turbulent flow region of $2300 < Re < 10^4$, the correlation of (Gnielinski, 1975) and, for larger Reynolds Numbers ($Re > 10^4$), the correlation of (Dittus and Boelter, 1930) is used to calculate the Nusselt number. If the high-pressure-side heat exchanger (gascooler) operates supercritically, the refrigerant-side heat transfer in the range of $2300 < Re < 10^4$ is calculated with the correlation of (Gnielinski, 1975), which is also used in the subcritical region, and single phase flow for the same range of the Reynolds number. If the high-pressure-side heat exchanger operates subcritically and the refrigerant passes the phase change process, the heat transfer is calculated with the correlation of (Cavallini et al, 2006). Additionally, for improving the correlation of (Cavallini et al, 2006) the approach of (Fujii and Watabe, 1987) was integrated as shown by (Kondou and Hrnjak, 2011). The heat transfer in the evaporators during the evaporation phase changing process is calculated with the correlation of (Gungor and Winterton, 1987).

Predicting hydraulic losses in the heat exchangers is complex during the phase change process. Two basic calculation methods for determining hydraulic losses in two-phase flow are shown, for instance, by (Wallis, 1969) and (Rohsenow et al, 1985). On one hand, the homogeneous calculation method applies, which assumes equal velocities for the liquid and vapor phase. On the other hand, the heterogeneous calculation method applies, in which the liquid and vapor phase have different flow velocities, so that slippage between the two phases can be considered. For calculations of hydraulic losses in the R-744 refrigerant cycle model, the homogeneous calculation method is used. According to the assumption of equal velocities of the liquid and vapor phase, the average density is calculated based on the vapor content of the assumed homogeneous flow (Baehr and Stephan, 2006). In addition, the dynamic viscosity of the assumed homogeneous flow is determined by the approach of (McAdams et al, 1942). The final calculation of the pressure drop in the two-phase flow is calculated by the explicitly formulated approximation of the Colebrook-White equation of (Swamee and Jain, 1976). The pressure drop in all refrigerant pipes and the turbulent flow region

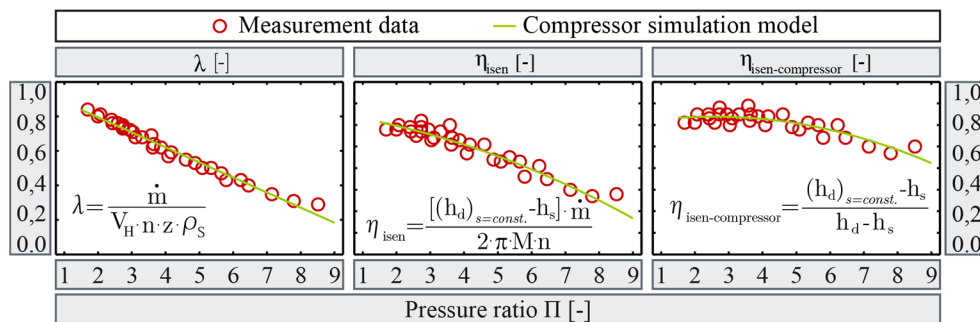


Figure 3. Comparison of implemented compressor model functions (green curves) for volumetric efficiency λ , isentropic efficiency η_{isen} and isentropic compressor efficiency $\eta_{isen-compressor}$ with measured data based points (red circles) at a constant compressor speed of $n=1000 \text{ min}^{-1}$ (measurement data and equation definition based on (Försterling, 2003)).

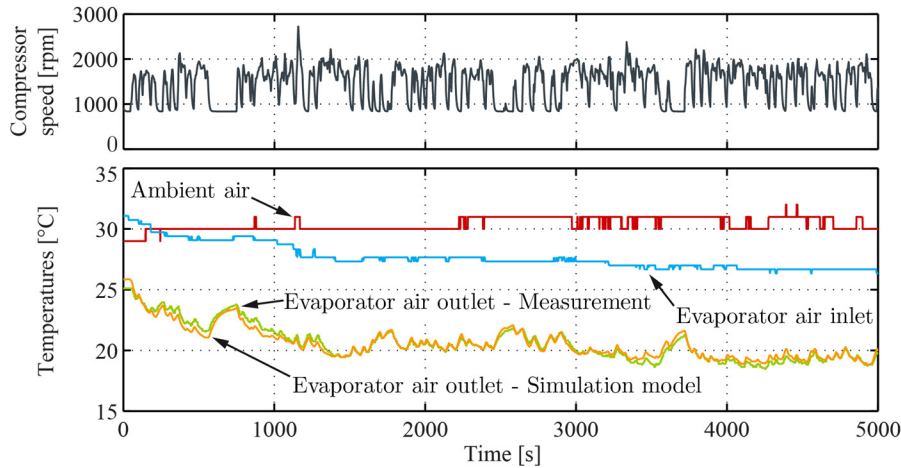


Figure 4. Top: compressor speed of simulation model based on measured engine speed. Bottom: measured ambient air temperature and measured air temperature at rooftop evaporator inlet (also used as evaporator inlet air temperature in simulation model) as well as comparison of measured and simulated air temperature at rooftop evaporator outlet.

($Re > 2300$) is calculated using the formulation of (Swamee and Jain, 1976) as well. In the laminar flow region ($Re < 2300$), the pressure drop in all refrigerant pipes is calculated with $\zeta = 64/Re$.

For modeling the compressor model, (TIL Suite, 2018) offers two basic modeling approaches: an efficiency-based compressor model and a physical-based compressor model. The compressor model used for the R-744 refrigerant cycle model follows the efficiency-based approach, as shown by (Försterling, 2003). Therefore, the compressor model is characterized by a volumetric efficiency λ , an isentropic efficiency η_{isen} and an isentropic compressor efficiency $\eta_{isen-Compressor}$. Figure 3 shows a comparison of the implemented model functions for the volumetric efficiency λ , isentropic efficiency η_{isen} and isentropic compressor efficiency $\eta_{isen-Compressor}$ with data points based on measurement data.

2.2 R-744 simulation model validation results

The previously described simulation model of the R-744 refrigerant cycle is calibrated and validated with measurement data of a vehicle measurement campaign. The following validation shows a section of this measurement campaign for a summer afternoon with an average ambient temperature of approximately 30°C.

For the calibration of the R-744 refrigerant cycle model, the previously specified heat transfer and pressure drop calculation at the refrigerant side were adapted with the aid of a constant calibration factor for each heat transfer and pressure drop calculation. As a result, the heat transfer and the pressure drop calculation at the gascooler, the IHX, the frontbox evaporator and rooftop evaporator were adapted based on constant calibration factors. At the refrigerant pipes, only the pressure drop calculation was adapted based on constant calibration factors in each pipe. Model calibration was performed using the complete closed refrigerant cycle model as presented in Figure 2, where measured data

were used as input values e.g. for the air temperature at the gascooler and both evaporator inlets as well as for compressor speed. For the model calibration, the deviation of the simulated refrigerant temperatures and pressures from measured refrigerant temperatures and pressures as well as the simulated and measured air

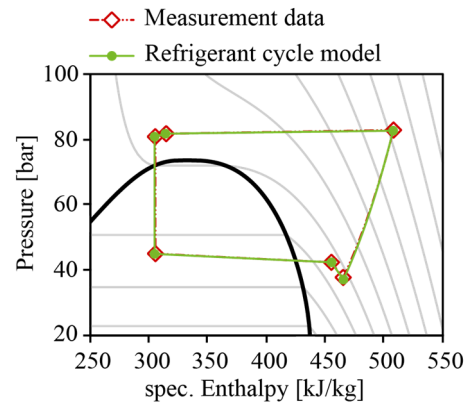


Figure 5. Comparison of calibration results of the R-744 refrigerant cycle model with measured data in ph-state diagram at $t = 2000s$.

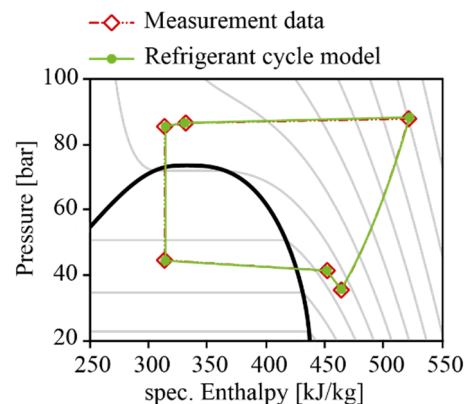


Figure 6. Comparison of calibration results of the R-744 refrigerant cycle model with measured data in ph-state diagram at $t = 3500s$.

temperatures at the outlet of the gascooler and evaporators were minimized using the method of the smallest error squares. The calculation of the air-side heat transfer and pressure drop were not modified.

The validation results of the calibrated R-744 refrigerant cycle model are shown in Figure 4, Figure 5 and Figure 6. Figure 4 shows the comparison of the measured and simulated air temperature at the outlet of the rooftop evaporator. The curves of the measured and simulated air temperatures fit together very well. The state diagrams in Figure 5 and Figure 6 show the refrigerant-side process compared to measured data at different times of the validation comparison (cf. Figure 4). This comparison of the simulated and measured process states shows a good matching. Therefore it can be said that the specific heat of the gascooler, evaporator and IHX as well as the refrigerant outlet temperature and pressure ratio of the compressor are simulated correctly. Finally, the modeled R-744 refrigerant cycle model shows a very good match to measured data.

Based on the very dynamic measurement data (as shown in Figure 4), the CPU time for integration of the R-744 refrigerant cycle model was more than 8 times faster than real time. To determine the CPU time, the R-744 refrigerant cycle model was simulated on a standard laptop.

3 Compressor capacity control method

The following section introduces and specifies an innovative method for implementing capacity adaptation in the refrigerant compressor of omnibus air conditioning systems. This innovative method consists, on one hand, of a compressor speed control by means of a two-speed pulley gearbox based on a planetary gearbox and, on the other hand, on a cylinder bank shutdown by suction gas interlock. The pulley gearbox based on a planetary gearbox is integrated in the compressor belt pulley, which was presented for automotive application by (Baumgart et al., 2006). With this integrated planetary gearbox, two transmission ratios ($i < 1$ and $i = 1$) can be implemented. Figure 7 shows

the design, schematic and operating principle of the pulley gearbox. Insofar as the brake is closed and the clutch is released (switch position I, $n_{\text{Planet carrier}} = 0$), the gearing between the ring gear and the sun gear generates a transmission ratio into higher speed ($i < 1$). If the brake is released and the clutch is closed, the gear unit rotates as one part (switch position II, $n_{\text{Planet carrier}} = n_{\text{Sun gear}}$, $i = 1$) and the gearbox runs without any friction losses. In this case, the refrigerant compressor is driven only by the transmission ratio of the belt drive. If both the brake and the clutch are released, the planetary gearbox is underdetermined and decouples the refrigerant compressor from the belt drive (switch position III), thus enabling the refrigerant compressor to be disconnected from the drive as before with a conventional magnetic clutch. The simulation model of the two-speed pulley gearbox converts the speed of the belt drive connected to the internal combustion engine (in subsystem 3, see Figure 1) to the compressor (in subsystem 5, see Figure 1) of the refrigerant cycle model presented in Figure 2. A control signal to the gearbox model activates or deactivates the speed conversion in the gearbox based on described switch positions and the Willis equation shown in Figure 7.

The cylinder bank shutdown by suction gas interlock is usually installed in the cylinder head of one cylinder or cylinder pair of the refrigerant compressor. Figure 8 illustrates the schematic and operating principle of the suction gas interlock. Insofar as no voltage is applied to the solenoid valve, the high pressure pass to the locking valve is closed and the spring pushes the locking valve into the upper valve seat. The connection between the suction chamber and suction gas line is open, and the refrigerant compressor operates at full capacity. When the solenoid valve is actuated, the access of the high pressure pass to the locking valve is opened, high pressure refrigerant flows above the locking valve and presses it into the lower valve seat. As a result, the connection between the suction chamber and suction gas line is blocked and the refrigerant compressor operates at reduced capacity. In the simulation model, the suction gas interlock is implemented by modifying the effective displacement volume ($V_{H,Z}$) of the

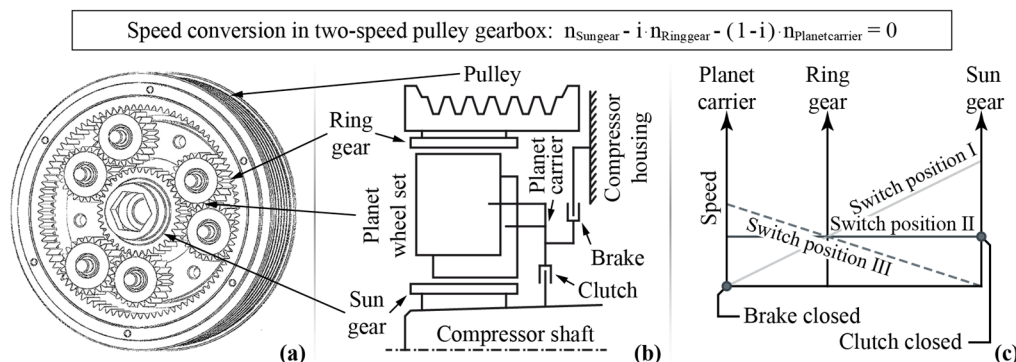


Figure 7. Design (a), schematic (b) and operating principle (c) of two-speed pulley gearbox. Two-speed pulley gearbox converts speed of the belt drive to refrigerant compressor shaft based on Willis equation above.

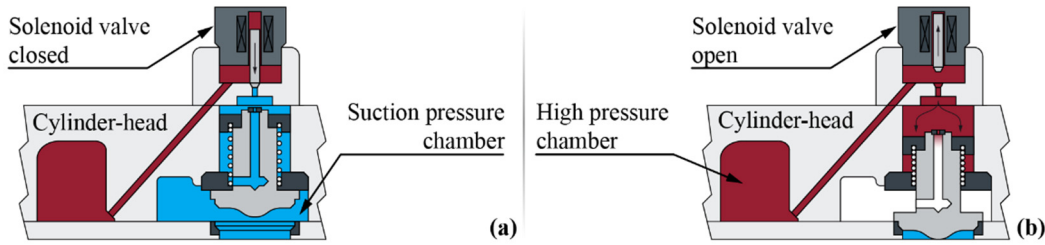


Figure 8. Schematic and operation principle of suction gas interlock: (a) inactive interlock, (b) active interlock. The suction gas interlock is integrated in the cylinder-head of a cylinder pair and locks the connection to the suction gas line.

refrigerant compressor model, see Figure 2. Because of this, volumetric efficiency and, as a result, isentropic and isentropic compressor efficiency are also affected, see equations in Figure 3.

Table 1. Transmission ratios for refrigerant compressor drive by two-speed pulley gearbox.

| Switch position | Transmission ratio i |
|-----------------|------------------------|
| I | 0.654 |
| II | 1.0 |

For the study within the vehicle simulation, Table 1 shows the selected transmission ratios for the refrigerant compressor drive with the two-speed pulley gearbox application. For the selection of the compressor drive, total transmission ratios of the belt drive and planetary gearbox, the conventional belt drive transmission ratio should be maintained. Selection of the second additional transmission ratio is based on a preliminary study (Kaiser et al., 2013). This second additional transmission ratio was selected in such a way that frequent shift operations of the two-speed pulley gearbox and activation of the suction gas interlock during normal operational speed changes of the internal combustion engine can be avoided in general driving mode. This is also intended to keep the superheat control from oscillating.

To adapt the refrigerant compressor speed, the two-speed pulley gearbox is controlled depending on the interior temperature (reflects the actual cooling demand). If the interior temperature reaches or exceeds the upper value of $t_{Set}+0.5K$, where t_{Set} is the interior set temperature, the two-speed pulley gearbox shifts into switch position I. If the interior temperature reaches or falls below the lower value t_{Set} , the two-speed pulley gearbox shifts into switch position II. Friction power losses with an active two-speed pulley gearbox in switch position I are calculated with a gear box efficiency of $\eta=0.96$ according to (Baumgart, 2010).

To adapt the refrigerant compressor capacity based on the cylinder bank shutdown, the suction gas interlock is controlled depending on the interior temperature as well. If the interior temperature reaches or falls below the lower value t_{Set} , where t_{Set} is the interior set temperature, the suction gas interlock application is activated. If the interior temperature reaches or exceeds

the upper value of $t_{Set}+0.5K$, the suction gas interlock application is deactivated. Based on the cylinder bank shutdown by suction gas interlock, the refrigerant compressor displacement volume can be controlled between 50% and 100%.

For the study within the vehicle simulation, the refrigerant compressor capacity control by means of the two-speed pulley gearbox application and the suction gas interlock application are used as follows: First, the refrigerant compressor speed is adapted by the two-speed pulley gearbox. Afterwards, the two-speed pulley gearbox is in switch position II, and the cylinder bank shutdown by suction gas interlock can be activated.

4 Virtual driving route test scenarios

Two climatically different driving scenarios were realistically modeled for the addressed research issues in the realm of bus air conditioning systems. For this, Figure 9 shows the selected driving routes in their respective map sections, which are dynamically driven through with the total vehicle model shown in Figure 1. Based on the geographic coordinates of these two driving scenarios, individual velocity and elevation profiles were calculated to describe the target state for the vehicle model driving simulation. Depending on the defined velocity profile and the geographic position, time-dependent representative ambient conditions were calculated based on a meteorological database (METEONORM, 2016; Remund et al., 2013) for the two driving route scenarios. For the presentation of an annual cross-section, transient ambient conditions are calculated for every 15th of the month in a representative year. Thus the ambient conditions include ambient air temperature, ambient air pressure, ambient relative humidity as well as direct and diffuse ambient solar radiation. Figure 9 shows the calculated ambient air temperature curves for the two driving route scenarios as an example. The numbered ambient air temperature curves represent the 15th of each numbered month. The background areas in gray represent the ambient temperature range in which the refrigerant circuit is not active with respect to the climate controller algorithms implemented. In this process, the refrigerant circuit is automatically switched off at $t_{amb} \leq 13^{\circ}C$ and automatically switched on at $t_{amb} \geq 15^{\circ}C$.

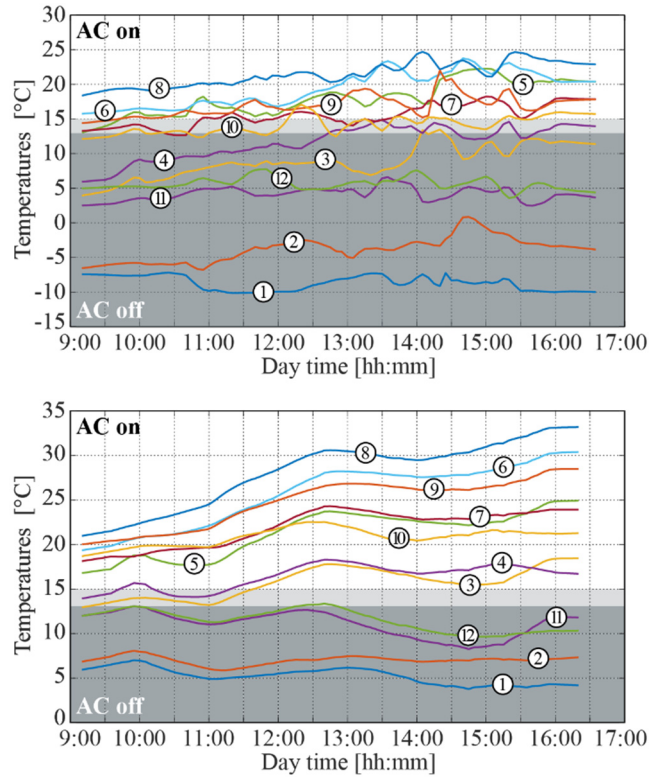
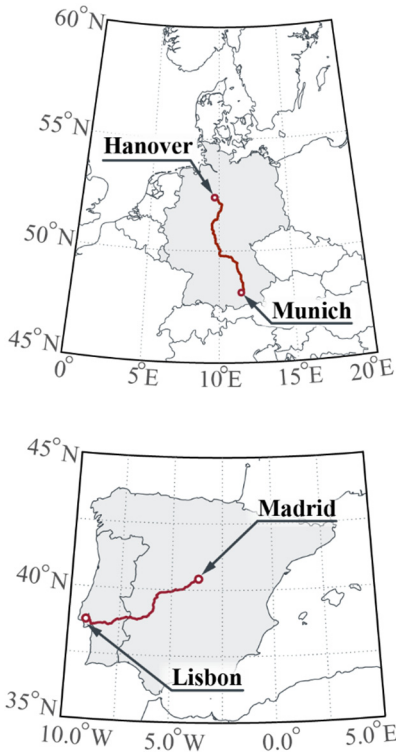


Figure 9. Realistically modeled driving route scenarios and corresponding transient ambient temperatures for every 15th of the month in a representative year based on a meteorological database. Driving route scenarios: Hanover to Munich (top), Lisbon to Madrid (bottom), numbered temperature curves: (1) represents 15th of January ... (12) represents 15th of December; gray temperature areas: $t_{amb} \leq 13^\circ\text{C}$ refrigerant cycle is off, $t_{amb} \geq 15^\circ\text{C}$ refrigerant cycle is on.

5 Results

This section presents the numerical results of the total vehicle simulation with application of the refrigerant compressor capacity control method described above.

Figure 11 shows the dynamic simulation results of

the total vehicle simulation with the R-744 refrigerant cycle for the virtual test driving route scenario from Lisbon to Madrid on a representative 15th of August. It shows the ambient boundary conditions including ambient air temperature, ambient air pressure, ambient relative humidity as well as direct and diffuse ambient solar radiation. It also shows the average interior

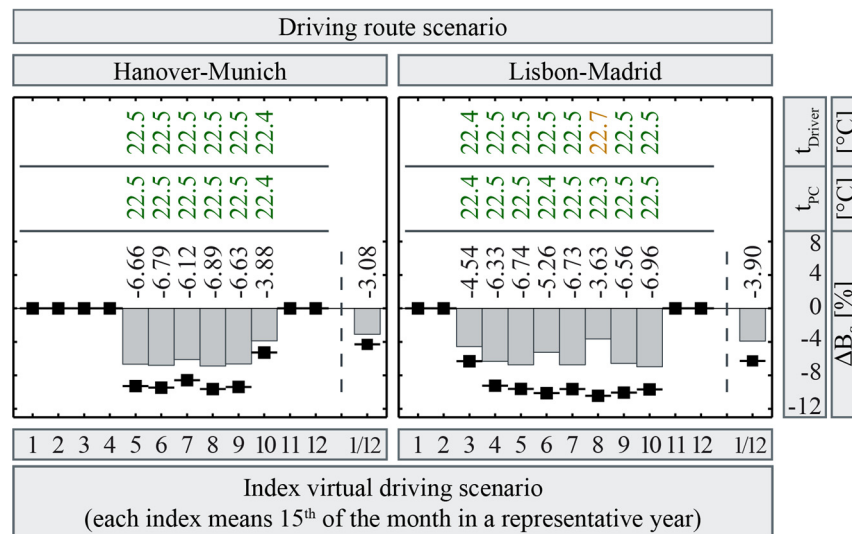


Figure 10. Numerical results of application combination of two-speed pulley gearbox and suction gas interlock for R-744-based air conditioning system. Shows the fuel consumption of each monthly driving scenario and, in the last column of each diagram, the average annual fuel consumption (1/12). Black boxes present the theoretical limit potential of fuel savings through operation of the air conditioning system. Set temperature for interior air temperature control in driver's workspace and passenger compartment is $t_{set} = 22.5^\circ\text{C}$.

temperature as well as the control behavior of the two-speed pulley gearbox and the suction gas interlock. Additionally it presents the refrigerant compressor speed and the compressor outlet refrigerant mass flow rate.

To evaluate the efficiency of the developed and investigated compressor capacity control method, vehicle fuel consumption is used for valuation purposes. This is done because vehicle fuel consumption includes all multivariable dependencies of the total vehicle model and of the investigated compressor capacity

control methods and their sensitivities to the total vehicle simulation model. The results of the change in vehicle fuel consumption are presented in relation to the reference vehicle model, which is validated against the absolute average fuel consumption of a typical coach, see (Kaiser, 2018).

Figure 10 shows the numerical results of the relative change in fuel consumption ΔB_s for the R-744 system with the combination of two-speed pulley gearbox and suction gas interlock depending on the two climatically different driving scenarios. The presented results are

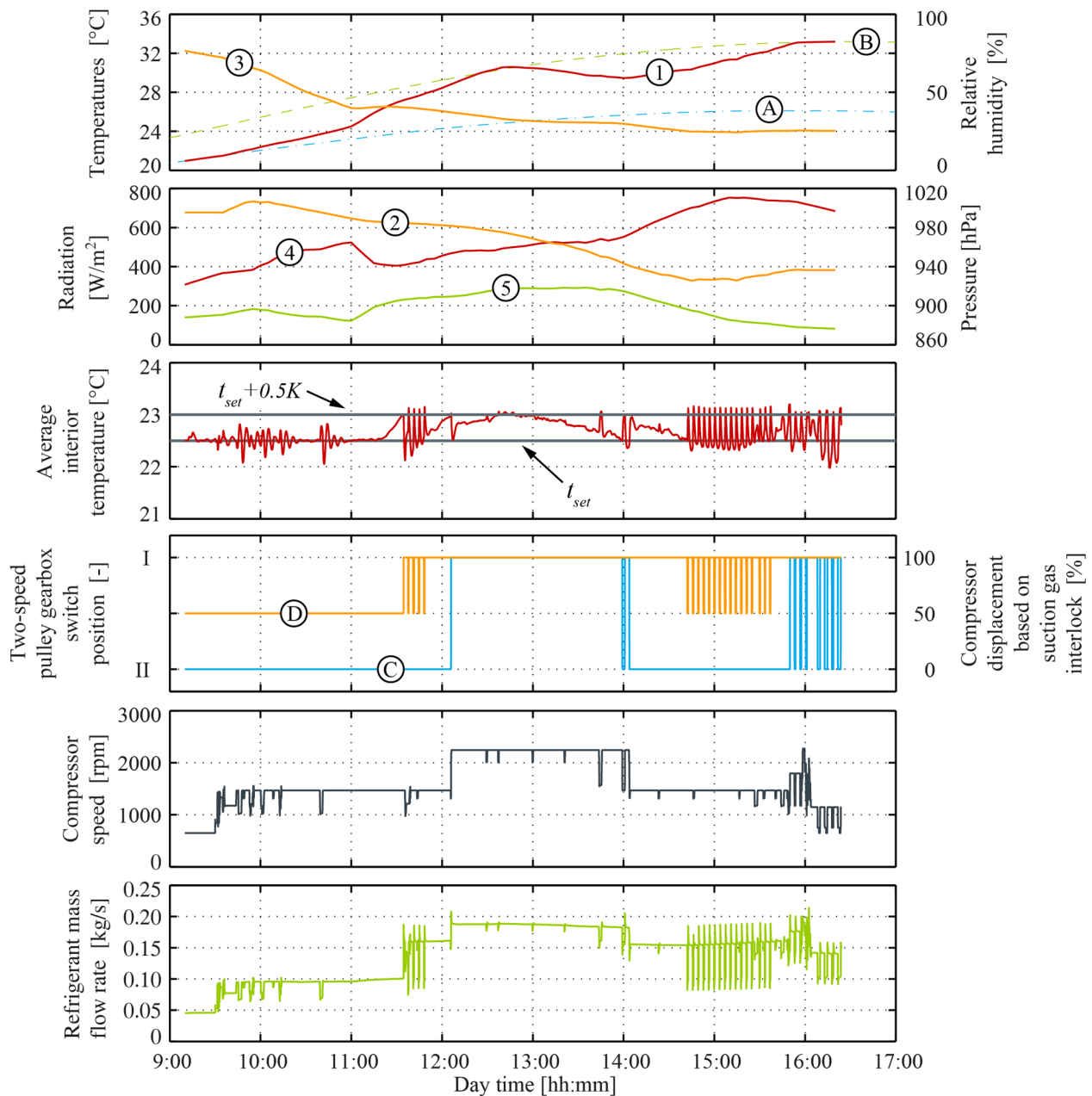


Figure 11. Example results of the total vehicle simulation with refrigerant compressor capacity control by two-speed pulley gearbox and suction gas interlock for virtual driving route scenario Lisbon-Madrid on a representative 15th of August (in accordance with index 8 of virtual driving route scenario). Shows ambient conditions of virtual driving route scenario: (1) ambient air temperature, (2) ambient air pressure, (3) ambient relative humidity as well as (4) direct and (5) diffuse ambient solar radiation; additionally (A) point of departure ambient air temperature, (B) point of destination ambient air temperature. Also shows average interior temperature; (C) control behavior of two-speed pulley gearbox and (D) suction gas interlock as well as refrigerant compressor speed and compressor outlet refrigerant mass flow rate.

shown in relation to the reference system simulations with the R-744-based air conditioning systems without the refrigerant compressor capacity control. In addition to the relative change in fuel consumption ΔB_S , the figures show the interior air temperature in the driver's workspace t_{Driver} as well as the average passenger compartment air temperature t_{PC} . Furthermore, the diagrams for the relative change in fuel consumption also include the theoretical limit potential of possible fuel savings achievable by operating the air conditioning system (black boxes). This theoretical limit potential is calculated within an additional reference vehicle simulation in which the use of the air conditioning system does not consume any energy.

The results in Figure 10 show a continuous reduction in fuel consumption. In all these driving scenarios, the cooling capacity produced by the R-744 reference air conditioning exceeds the actual cooling demand. As a result, the compressor capacity control by two-speed pulley gearbox and suction gas interlock reduce the refrigerant compressor capacity as shown in Figure 11. In this manner, the cooling capacity is more closely matched to the actual cooling demand and fuel consumption is reduced. The results in Figure 10 also show that the interior air temperature in the passenger compartment still conforms to the set air temperature of $t_{\text{Set}}=22.5^\circ\text{C}$ even when the compressor capacity is adapted by the two-speed pulley gearbox and suction gas interlock. Fuel consumption, and thus the indirect emissions mentioned above through use of the air conditioning system, can be significantly reduced with the presented refrigerant compressor capacity control. The coach's total fuel consumption is reduced by an average of 6.2% in the Hanover-Munich driving scenario. In the Lisbon-Madrid driving scenario, total fuel consumption is reduced by an average of 5.8%.

The CPU time for integration of the total vehicle simulation model, including the R-744 refrigerant cycle as well as the compressor capacity control by two-speed pulley gearbox and suction gas interlock, was about 2 times faster than the real time. This specific simulated total vehicle model has 830 continuous time states and 28012 time-varying variables. The translated model has 8 linear equations, where the largest linear equation system has size 8. Further the translated model has 53 nonlinear equations, where the largest nonlinear equation system has size 3. Model translation and manipulation was done using Dymola.

6 Conclusion

The refrigerant cycle in conventional omnibus HVAC systems has a significant influence on fuel consumption and, as a result, on so-called indirect emissions. In addition, direct emissions occur in the refrigerant cycle caused by unintended leakage of refrigerant. To achieve reduced indirect and direct emissions, a natural refrigerant, CO₂-based (R-744) air conditioning system

with compressor capacity control is a purposeful solution. For this reason a detailed model of an R-744 refrigerant cycle based on the Modelica library (TIL Suite, 2018) was developed and validated. Furthermore, an innovative refrigerant compressor capacity control method based on a combination of speed control by two-speed pulley gearbox and suction gas interlock was presented. To study the R-744-based air conditioning in combination with the presented compressor capacity control, two different realistically modeled driving route scenarios were shown, which were virtually and dynamically driven through using a total vehicle model of a coach. In comparison to an R-744-based omnibus HVAC system, which only reduces unintended direct emissions, the compressor capacity control by combination of two-speed pulley gearbox and suction gas interlock additionally reduces indirect emissions through significant fuel savings. As a result, the coach's total fuel consumption can be reduced by an average of 6% in the considered virtual test driving scenarios.

The presented R-744 refrigerant cycle model has 391 continuous time states and 19740 time-varying variables. The translated model of the R-744 refrigerant cycle has 2 linear equations, where the linear equation system is size 2 and 3. Furthermore, the translated model has 20 nonlinear equations, where each nonlinear equation system has size 1. The total vehicle simulation model including the R-744 refrigerant cycle as well as the compressor capacity control by two-speed pulley gearbox and suction gas interlock has 830 continuous time states and 28012 time-varying variables. The translated model has 8 linear equations, where the largest linear equation system has size 8. Furthermore, the translated model has 53 nonlinear equations, where the largest nonlinear equation system has size 3. Model translation and manipulation was done using Dymola.

The CPU time for integration of the standalone R-744 refrigerant cycle model was more than 8 times faster than real time. The CPU time for integration of the total vehicle simulation model, including the R-744 refrigerant cycle as well as the compressor capacity control by two-speed pulley gearbox and suction gas interlock, was about 2 times faster than real time. To determine the CPU time, the R-744 refrigerant cycle model and the total vehicle simulation model were simulated on a standard laptop.

References

- Baehr, H. D., Stephan, K. (2006): *Wärme- und Stoffübertragung*. Springer Verlag, 2006.
- Baumgart, R.; Tenberge, P.; Webner, M.; Gebhardt, J. (2006): Riemenscheibe mit integriertem Getriebe zur Drehzahlregelung des Klimakompressors. *Wärmemanagement des Kraftfahrzeuges V*: 243-255, Expert-Verlag, 2006.

- Baumgart, R. (2010): *Reduzierung des Kraftstoffverbrauches durch Optimierung von Pkw-Klimaanlagen*. Doctoral dissertation, Technische Universität Chemnitz, 2010.
- Cavallini, A.; del Col, D.; Doretti, L.; Matkovic, M.; Rossetto, L.; Zilio, C. (2006): Condensation in horizontal smooth tubes. A new heat transfer model for heat exchanger design. *Heat Transfer Engineering*, 27(8): 31–38, Taylor & Francis, 2006.
- Dittus, F. W.; Boelter, L. M. K. (1930): Heat Transfer in Automobile Radiators of the Tubular Type. *Publications on Engineering*, 2: 443-461, University of California at Berkeley, 1930.
- Försterling, S. (2003): *Vergleichende Untersuchung von CO₂-Verdichtern in Hinblick auf den Einsatz in mobilen Anwendungen*. Doctoral dissertation, Technische Universität Braunschweig, 2003.
- Fujii, T.; Watabe, M. (1987): Laminar Film Condensation in the Subcritical Region – Gravity Controlled Condensation. *JSME Trans*, 53: 1801–1806, JSEM – Japan Society of Mechanical Engineers, 1987.
- Gnielinski, V. (1975): Neue Gleichungen für den Wärme- und den Stoffübergang in turbulent durchströmten Rohren und Kanälen. *Forschung im Ingenieurwesen – Engineering Research*, 41(1): 8 – 16, Springer-Verlag, 1975.
- Gräber, M.; Kosowski, K.; Richter, C.; Tegethoff, W. (2010): Modelling of heat pump with an object-oriented model library for thermodynamic systems. *Mathematical and Computer Modelling of Dynamical Systems*, 16(3): 195–209, Taylor & Francis, 2010.
- Gungor, K. E.; Winterton, R. H. S. (1987): Simplified General Correlation for Saturated Flow Boiling and Comparison of Correlation with Data. *Chemical Engineering Research and Design*, 65(2): 148–156, Elsevier Science B.V., 1987.
- Haaf, S. (1988): Wärmeübertragung in Luftkühlern. *Handbuch der Kältetechnik*: 6(B), Springer-Verlag, 1988.
- Hebeler, M.; Ebeling, P.; Tegethoff, W.; Köhler, J.: Exhaust Waste Heat Recovery for Intercity Bus Climatisation using Rankine Technology with Focus on Topology Design. *2nd ETA Conference*, IAV Automotive Engineering, 2018.
- Kaiser, C., Baumgart, R., Aurich, J., Tegethoff, W., Köhler, J. (2013): Konzepte für die Reduzierung des Kraftstoffverbrauches von Omnibusklimaanlagen. *12. Internationale Fachtagung Nutzfahrzeuge*: 269-284, VDI-Verlag, 2013.
- Kaiser, C. (2018): *Untersuchungen zur Effizienz- und Leistungsverbesserung von Omnibusklimaanlagen*. Unpublished doctoral dissertation, Technische Universität Braunschweig, 2018.
- Kondou, C.; Hrnjak, P. (2011): Heat Rejection from R744 Flow Under Uniform Temperature Cooling in a Horizontal Smooth Tube around the Critical Point. *International Journal of Refrigeration*, 34: 1293– 301, Elsevier Science B.V., 2011.
- McAdams, W. H.; Woods, W. K.; Heroman, L. C. (1942): Vaporization inside horizontal tubes-II-benzene-oil mixtures. *Trans. ASME*, 64(3): 193–200, American Society of Mechanical Engineers, 1942.
- Meise, S.; Kaiser, C.; Engel, P.; Lemke, N.; Köhler, J.: R-744-Ejektor-Wärmepumpe für elektrische Gelenkbusse. *Deutsche Kälte-Klima-Tagung*, DKV e.V., 2018.
- METEONORM (2016): *Software for the determination of worldwide weather data*. METEOTEST, Version: 7.1.10 [Computer Software], 2016.
- Remund, J.; Müller, S.; Kunz, S.; Huguenin-Landl, B.; Schmid, C.; Schilter, C. (2013): *METEONORM – Global Meteorological Database. Handbook I/II*. METEOTEST, 2013.
- Richter, C. C. (2008): *Proposal of New Object-Oriented Equation-Based Model Libraries for Thermodynamic Systems*. Doctoral dissertation, Technische Universität Braunschweig, 2008.
- Rohsenow, M. W.; Hartnett, P. J.; Ganic, E. N. (1985): *Handbook of Heat Transfer Fundamentals*. McGraw-Hill, 1985.
- Schulze, C. W. (2013): *A Contribution to Numerically Efficient Modelling of Thermodynamic Systems*. Doctoral dissertation, Technische Universität Braunschweig, 2013.
- Swamee, P. K.; Jain, A. K. (1976): Explicit Equations for Pipe-Flow Problems. *Journal of the Hydraulics Division*: 102(5): 657–664, ASCE – American Society of Civil Engineers, 1976.
- Tegethoff, W.; Schulze, C.; Gräber, M.; Huhn, M.; Stulgies, N.; Kaiser, C.; Loeffler, M. (2011): *TEMO: Thermische echtzeitfähige Modelle*. Bundesministerium für Bildung und Forschung (BMBF), 2011.
- TIL Suite (2018): *Software package for the simulation of thermal systems*. TLK-Thermo GmbH, Version 3.4 [Computer Software], 2018.
- Wallis, G. B. (1969): *One-dimensional Two-phase Flow*. MCGraw-Hill, 1969.

Nomenclature

| | | |
|--------------|--|--------------------|
| Nu | Nusselt number | (-) |
| Re | Reynolds number | (-) |
| η | Efficiency | (-) |
| λ | Volumetric efficiency | (-) |
| ζ | Pressure loss coefficient | (-) |
| h | Specific enthalpy | (J/kg) |
| i | Transmission ratio | (-) |
| \dot{m} | Mass flow rate | (kg/s) |
| M | Torque | (Nm) |
| n | Speed | (s ⁻¹) |
| t_{amb} | Ambient air temperature | (°C) |
| t_{Set} | Set value of interior air temperature | (°C) |
| t_{Driver} | Air temperature in driver's workspace | (°C) |
| t_{PC} | Air temperature in passenger compartment | (°C) |
| ΔB_S | Relative difference in fuel consumption | (%) |
| IHX | Internal heat exchanger | - |
| V_H | Displacement volume | (m ³) |
| VLE | Vapor-liquid equilibrium | - |
| z | Number of cylinders | (-) |

Control Description Language

Michael Wetter Milica Grahovac Jianjun Hu

Lawrence Berkeley National Laboratory
Energy Technologies Area
Building Technology and Urban Systems Division
Berkeley, CA, USA
{mwetter, mgrahovac, jianjunhu}@lbl.gov

Abstract

Properly designed and implemented building control sequences can significantly reduce energy consumption. However, there is currently no process with supporting tools that allows the assessment of the performance of different control sequences, export the control sequences in a vendor-neutral format for cost estimation and for implementation on a building automation system through machine-to-machine translation, and reuse the sequences for verification during commissioning.

This paper describes a Control Description Language (CDL) that we developed to create such a process. For CDL, we selected a subset of Modelica that allows a convenient representation of control sequences, simulation of the control sequence coupled to a building energy model, and development of translators from CDL to building automation systems. To aid in the development of such translators, we created a translator from CDL to a JSON intermediate format. In future work, we seek to work with building control providers to develop translators from CDL to commercial building automation systems.

Through a case study, we show that CDL suffices for simulation-based performance assessment of two ASHRAE-published control sequences for a variable air volume flow system of an office building. Moreover, the case study showed that merely due to differences in the control sequences, annual HVAC energy use was reduced by 30%. This difference is larger than the accuracy required when comparing different HVAC systems, thereby questioning the current practice of idealizing control sequences in building energy simulations, and demonstrating the importance of ensuring that the control sequence used during design simulations corresponds to the control sequence that will be implemented in the real building.

Keywords: controls, buildings, HVAC

1 Introduction

The building control industry has a standard for data communication called BACNet that is supported by all major control vendors (ASHRAE, 2004). However, there is no standard for expressing the control logic, despite the situation that control is often not implemented as speci-

fied during design, and the savings potential due to better control sequences is significant but not widely realized. The purpose of this paper is to describe a first implementation of a language with the intent to develop a standard for expressing building control sequences. This standard should support the mechanical designer in developing and testing control sequences within building energy simulations, and exporting these sequences to create unambiguous specifications for the control provider. It should support control providers in cost-estimation and in implementation of the control sequence on their control platform through machine-to-machine translation, and it should support the commissioning agent when verifying that the implemented control sequence meets the original specification.

It is generally recognized that properly designed and implemented control sequences can reduce energy consumption around 20% to 30% (Fernandez et al., 2017). Implementation errors in control sequences are in particular common in large buildings as they typically have built-up heating, ventilation and air-conditioning (HVAC) systems that require custom control sequences. The need for correct design and implementation of energy-saving and load-shifting control sequences is increasing because more stringent demands on energy savings and energy flexibility for the grid leads to increased complexity of control sequences.

For built-up HVAC systems, the current process is generally as follows: The mechanical engineer writes the control sequence in English language. This typically involves copying and adapting a part of the control sequence from similar projects. The document is then sent to a control provider. The control provider uses this English language description for cost estimation, and later for implementation. The control provider typically implements the control sequence by combining parts of sequences from previous projects that appear to have a similar controls intent. During commissioning, the commissioning agent conducts a limited number of tests to verify that the operation conforms to the commissioning agents' understanding of the control sequence.

The quality of the English language descriptions that are underlying this process varies largely. Our observation

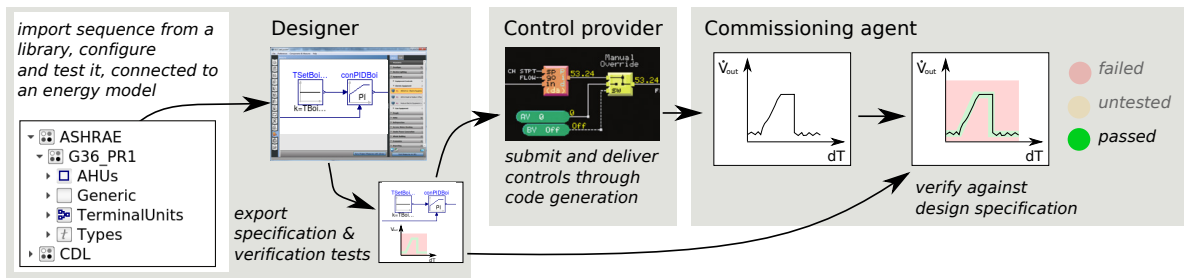


Figure 1. Overview of process for control sequence design, export of a specification, implementation on a control platform and verification against the specification.

is that best-in-class sequence specifications are often ambiguous, leave considerable room for interpretation, and may miss part of the sequence. This is not surprising because the control sequences are rather complex, because the English language representation aims to convey the control intent rather than how to realize it, and the mechanical engineer who wrote the control sequences is neither trained in, nor reimbursed for, the implementation of control sequences. As a consequence, such control specifications are not executable and hence do not allow for formal testing.

To realize the energy savings potential of building control sequences, we are working on developing tools and a process that will allow a mechanical engineer to select and adapt a control sequence from a library of sequences, connect it to a simulation model of the HVAC and building, test the performance of the control sequence coupled to the simulation model of the HVAC system and the building, export the control sequence in a control-vendor neutral format that allows control providers to conduct cost estimates and ultimately implement the sequence on their control platform through machine-to-machine translation. Figure 1 shows an overview of such a design flow.

To enable such a design flow, we are developing a language that we call Control Description Language (CDL), whose description is the main subject of this paper. We are also working on a project called "Spawn of EnergyPlus" that redesigns EnergyPlus so that it supports this process (see <https://lbl-srg.github.io/soep/>).

CDL needs to satisfy these high level requirements:

- It must be independent of any control-vendor specific platform.
- It must be declarative to facilitate its translation to other languages.
- It must be possible to simulate controls expressed in the language within an annual building energy simulation.
- It must be deterministic, e.g., for given inputs and states, different implementations of sequences expressed must yield the same output and state updates (within the precision of ordinary differential equation solvers that may integrate PID controllers).
- It should be possible to translate the sequence to a

variety of building control platforms.

- It must allow identification of cyclic graphs that would require iterative solutions and hence are not suited for implementation in building automation systems.

Related work in our application domain includes the following: Husaunndee et al. (1997) developed a MATLAB/Simulink-based toolbox of models of HVAC components and plants for the design and test of control systems called SIMBAD. SIMBAD has been used for testing and emulation of building control sequences, and is commercially distributed by CSTB France. Bonvini and Leva (2012) developed an industrial control library in Modelica that contains a variety of blocks with the intent to allow modelers to replicate industrial control sequences, including vendor-specific peculiarities. Yang et al. (2010) developed a tool chain that maps Simulink and Modelica models into an intermediate format, and then refined it for implementation in distributed controllers. Our approach borrows from their methodology in that we also use an intermediate format and restrict the language to make such a translation possible. Schneider et al. (2017) implemented a Modelica library with standardized control functions for building automation. They use control functions from VDI 3813-2:2011 and state graph representations from VDI 3814-6:2009. Our approach differs from their work as they document the control sequence using Unified Modeling Language (UML) class and activity diagrams. Also, they used semantic control connectors, which they subsequently removed for version 1.0.0. To generate English language representations together with a process diagram, Automated Logic Control developed CtrlSpecBuilder (ALC, 2018). CtrlSpecBuilder allows mechanical engineers to select the desired control functionality by answering a set of questions. The software then outputs a Microsoft Word document that specifies the control sequence in a vendor-neutral way together with a process diagram. Our approach differs from Bonvini and Leva (2012), Yang et al. (2010) and Schneider et al. (2017) in that we use elementary control blocks that form a basic library of control functions, and simple composition rules that we believe suffice for composing building control sequences.

As of this writing, we implemented CDL, used it

to implement control sequences that were developed by a project conducted for the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE), tested these sequences in simulation, and developed an export program that converts the CDL representation to a JSON and an English language representation. Work in progress and not reported in this paper includes the use of CDL to compare a simulated versus an actual building control system response. Also ongoing are discussions with control providers to see if they can prototype a translator from CDL to their commercial product line. Based on this feedback, the CDL language may further evolve.

2 Discussion of the Target Platform

To put this work in context, one has to recognize that building control systems largely vary in how they implement control sequences. Commercial products range from textual languages that combine the functionality of FORTRAN with programming structures similar to BASIC (Siemens, 2000) to graphical block-diagram languages where blocks can be used from a library and new blocks can be provided in a component-oriented programming language similar to Java or C# (Thomas, 2016). Furthermore, different building control systems use different native data types; some allow boolean signals to take on `true` or `false` only, while others also allow the value of `null`. Also, control sequences often contain proprietary algorithms, such as the computation of the start time for a warm-up after a room temperature setback. Furthermore, specification for the programming languages are hard if not impossible to find for many systems. Thus, the space of target platforms to which our language will need to be translated is heterogeneous and often proprietary. We therefore only intent to translate CDL to building automation systems, but do not attempt to translate a particular control implementation to CDL.

Control providers typically also include blocks for communication with hardware, and for sending messages to the operator, such as through logging, sending email, or displaying a value in an operator workstation. For example, Contemporary Controls' Sedona platform contains a block called `CControls_BASR8M_Platform` that advises the programmer how much usable memory is available for application programming, a block to monitor the execution time of a Sedona logic (`ScanTim`) and blocks to communicate with BACNet or with web pages (Contemporary Controls, 2017). CDL does not attempt to support such specialized blocks. Rather, the intention of CDL is to support the declaration of the control logic in a vendor neutral way. This is also required because during the design of a building, the control provider may not yet be known and thus the specification should be independent of any control product line. Code that provides input/output functionality with hardware or web services will need to be added when a CDL-conformant con-

trol sequence specification is implemented on a particular control platform.

3 CDL Language

We will now describe the Control Description Language (CDL). To develop CDL, we identified a small subset of basic control functionalities that will need to be provided, together with rules that prescribe how to compose sequences and rules that prescribe the mathematical behavior of these basic control functionalities and composite sequences. Specifically, we formulated CDL as a block diagram language that consists of the following elements:

- Permissible data types.
- Elementary control blocks, each of which encapsulates an elementary calculation performed on a signal in a control sequence, such as a block that adds two signals and outputs the sum.
- Input and output connectors through which these blocks receive values and send values.
- Syntax to specify
 - how to instantiate control blocks and assign values to parameters, such as a proportional gain,
 - how to connect inputs of blocks to outputs of other blocks,
 - how to document blocks,
 - how to add annotations, such as for graphical rendering of blocks and their connections, and
 - how to specify composite blocks.
- A model of computation that describes when blocks are executed and when outputs are assigned to inputs.

The following sections further explain these elements.

3.1 Syntax

In order to use CDL with building energy simulation programs, and to not invent yet another language with a new syntax, we selected a subset of the Modelica 3.3 specification for the implementation of CDL (Modelica Association, 2012). The selected subset is needed to instantiate classes, assign parameters, connect objects and document classes. This subset is fully compatible with Modelica, e.g., no other information that violates the Modelica Standard has been added, thereby allowing users to view, modify and simulate CDL-conformant control sequences with any Modelica-compliant simulation environment.

To simplify the support of CDL for tools and control systems, the following Modelica keywords are not supported in CDL: `extends`, `redeclare` and `constrainedby`, `inner` and `outer`.

Also, the following Modelica language features are not supported in CDL:

1. `Clocks`, as the use of clocks would complicate translation to building automation systems that often distribute the control sequences to different field devices.
2. `algorithm` sections, because the elementary building blocks are black-box models as far as CDL

is concerned and thus there is no need to support algorithm sections.

- initial equation and initial algorithm sections, because these are not needed when composing sequences using the elementary building blocks explained in Section 3.4.

3.2 Permissible Data Types

The basic data types are, in addition to the elementary building blocks, parameters of type `Real`, `Integer`, `Boolean`, `String`, and `enumeration`. All specifications in CDL are declarations of blocks, instances of blocks, or declarations of type `parameter`, `constant`, or `enumeration`. Variables are not allowed.¹ The declaration of such types is identical to the declaration in Modelica.

Each of these data types, including the elementary building blocks, can be a single instance or one-dimensional array. Array indices shall be of type `Integer` only. The first element of an array has index 1. An array of size 0 is an empty array. `enumeration` or `Boolean` data types are not permitted as array indices.

3.3 Encapsulation of Functionality

All computations are encapsulated in a `block`. Blocks expose parameters, and they expose inputs and outputs using connectors.

Blocks are either *elementary building blocks* (see Section 3.4) or *composite blocks* (see Section 3.9).

3.4 Elementary Building Blocks

The CDL library contains elementary building blocks that are used to compose control sequences. The functionality of elementary building blocks, but not their implementation, is part of the CDL specification. Thus, in the most general form, elementary building blocks can be considered as functions that for given parameters p , time t and internal state $x(t)$, map inputs $u(t)$ to new values for the outputs $y(t)$ and states $x'(t)$, e.g.,

$$(p, t, u(t), x(t)) \mapsto (y(t), x'(t)). \quad (1)$$

Control providers who support CDL need to be able to implement the same functionality as is provided by the elementary CDL blocks. CDL implementations are allowed to use a different implementation of the elementary building blocks, because the implementation is language specific. However, implementations shall have the same inputs, outputs and parameters, and they shall compute the same response for the same value of inputs and state variables.

Users are not allowed to add new elementary building blocks. Rather, users can use them to implement composite blocks.

The elementary building blocks are implemented in subpackages of the package `CDL`. For each elementary

¹Variables are used in the elementary building blocks, but these can only be used as inputs to other blocks if they are declared as an output.

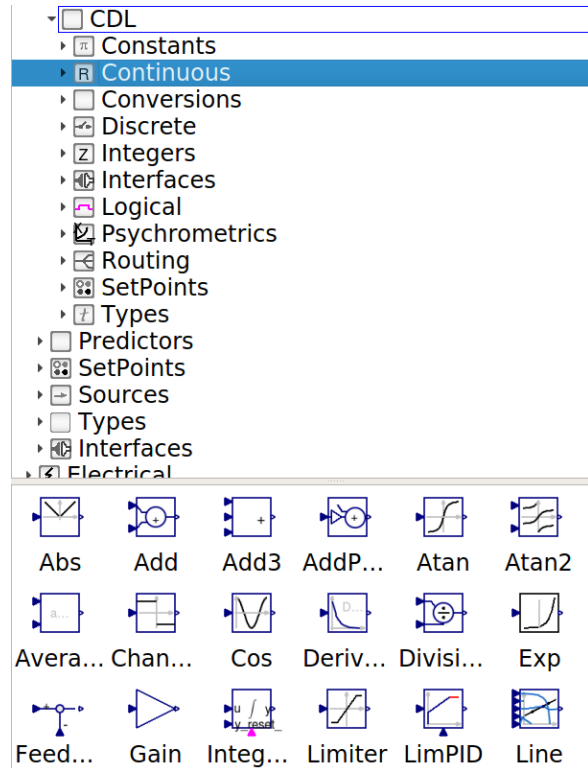


Figure 2. Screenshot of CDL library.

building block, there is an example that demonstrates its use.

An actual implementation of an elementary building block looks as follows, where we omitted the annotations that are used for graphical rendering:

```

block AddParameter
  "Output the sum of an input plus a
  parameter"
  parameter Real p "Value to be added";
  parameter Real k "Gain of input";
  Interfaces.RealInput u
  "Connector of Real input signal";
  Interfaces.RealOutput y
  "Connector of Real output signal";
equation
  y = k*u + p;
  annotation(Documentation(info("
  <html>
  <p>
  Block that outputs ... [omitted]
  </p>
  </html>"));
end AddParameter;

```

3.5 Instantiation

The instantiation of blocks is identical to Modelica. In the assignment of parameters, calculations are allowed. For example, a hysteresis block could be configured as follows

```

parameter Real pRel(unit="Pa") = 50
  "Pressure difference across damper";

```

```

CDL.Logical.Hysteresis hys (
  uLow = pRel-25,
  uHigh = pRel+25)
"Hysteresis for fan control";

```

Instances can conditionally be removed by using an `if` clause. This allows, for instance, to have a single implementation of an economizer enable/disable control sequence that can be configured to optionally take the specific enthalpy as an input signal. An example code snippet is

```

parameter Boolean use_enthalpy = true
"Set to true to evaluate outdoor air
enthalpy in addition to temperature"
;
CDL.Interfaces.RealInput hOut
if use_enthalpy
"Outdoor air enthalpy";

```

3.6 Connectors

Blocks expose their inputs and outputs through input and output connectors. The permissible connectors are implemented in the package `CDL.Interfaces`, and are `BooleanInput`, `BooleanOutput`, `DayTypeInput`, `DayTypeOutput`, `IntegerInput`, `IntegerOutput`, `RealInput` and `RealOutput`. `DayType` is an enumeration for working day, non-working day and holiday.

3.7 Connections

Connections connect input to output connectors. For scalar connectors, each input connector of a block needs to be connected to exactly one output connector of a block. For vectorized connectors, each (element of an) input connector needs to be connected to exactly one (element of an) output connector. Vectorized input connectors can be connected to vectorized output connectors using one connection statement, provided that they have the same number of elements.

Connections are listed after the instantiation of the blocks in an equation section. The syntax is

```

connect (port_a, port_b) annotation (...);

```

where `annotation(...)` is used to declare the graphical rendering of the connection (see Section 3.8). The order of the connections and the order of the arguments in the `connect` statement does not matter.

Signals shall be connected using a `connect` statement; assigning the value of a signal in the instantiation of the output connector is not allowed.

3.8 Annotations

Annotations follow the same rules as described in the following sections of the Modelica 3.3 Specification:

- §18.2 Annotations for Documentation.
- §18.6 Annotations for Graphical Objects, with the exception of
 - §18.6.7 User input, and

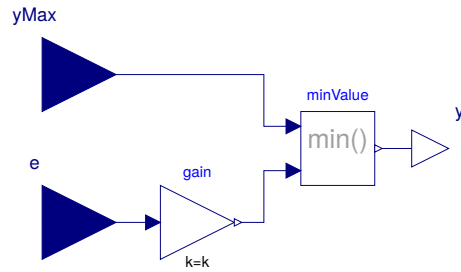


Figure 3. Example of a composite control block that outputs $y = \min(ke, y_{max})$, where k is a parameter.

- §18.8 Annotations for Version Handling.

Hence, for CDL, annotations are primarily used to graphically visualize block layouts and input and output signal connections, and to declare vendor annotations (see § 18.1 in Modelica 3.3 Specification).

3.9 Composite Blocks

CDL allows building composite blocks such as shown in Figure 3. Composite blocks are needed to preserve grouping of control blocks and their connections, and are needed for hierarchical composition of control sequences.

Composite blocks can contain other composite blocks. Each composite block shall be stored on the file system under the name of the composite block with the file extension `.mo`, and with each package name being a directory. The name shall be an allowed Modelica class name. Appendix A shows how to declare the block shown in Figure 3.

3.10 Model of Computation

CDL uses the synchronous data flow principle and the single assignment rule, which are defined below. The definition is adopted from and consistent with the Modelica 3.3 Specification § 8.4, and is as follows:

1. All variables keep their actual values until these values are explicitly changed. Variable values can be accessed at any time instant.
2. Computation and communication at an event instant does not take time.
3. Every input connector shall be connected to exactly one output connector.

In addition, the dependency graph from inputs to outputs that directly depend on inputs shall be directed and acyclic. I.e., connections that form an algebraic loop are not allowed.

3.11 Inferred Properties

CDL has sufficient information for tools that process CDL to generate for example point lists that list all analog temperature sensors, or to verify that a pressure control signal is not connected to a temperature input of a controller. Some, but not all, of this information can be inferred from the CDL language described above.

Note that none of this information affects the computation of a control signal. Rather, it can be used for example

to facilitate the implementation of cost estimation tools, or to detect incorrect connections between outputs and inputs.

To avoid that signals with physically incompatible quantities are connected, tools that parse CDL can infer the physical quantities from the `unit` and `quantity` attributes.

Therefore, tools that process CDL can infer the following information:

- Numerical value: Binary value (which in CDL is represented by a `Boolean` data type), analog value (which in CDL is represented by a `Real` data type) mode (which in CDL is presented by an `Integer` data type or an enumeration, which allow for example encoding of the ASHRAE Guideline 36 Freeze Protection which has 4 stages).
- Source: Hardware point or software point.
- Quantity: such as Temperature, Pressure, Humidity or Speed.
- Unit: Unit and preferred display unit. The use of display unit allows for example a control vendor to use the same sequences in North America displaying IP units, and in the rest of the world displaying SI units.

4 Control Sequence Implementation

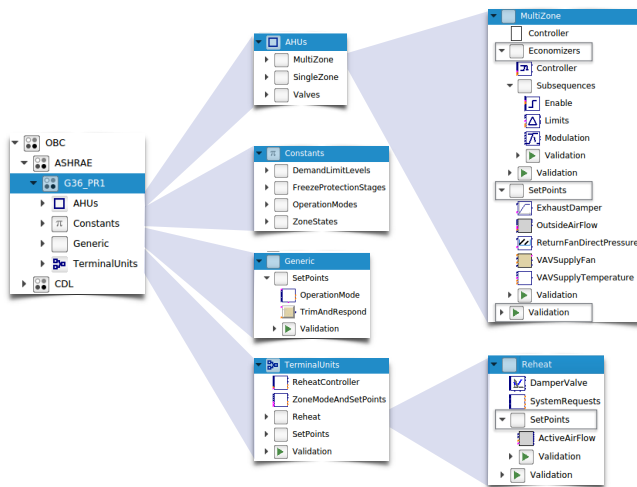


Figure 4. Overview of the ASHRAE Guideline 36 package implemented in the Modelica Buildings library 5.0.0

To test CDL, we used it to implement control sequences for variable air volume flow systems as specified in ASHRAE Guideline 36, public review draft 1 (ASHRAE, 2016). Figure 4 shows an overview of the package structure. The implementation is structured hierarchically into packages for air handler units, into constants that indicate operation modes, into generic sequences such as for a trim and respond logic, and into sequences for terminal units. For every sequence, there is a validation package that illustrates its use.

For implementation of these sequences, we had to make the following main design decisions:

For the PID controller, we used the same implementation as is used in the Modelica Buildings library. This implementation is identical to the one from the Modelica Standard Library, except that it adds an option to reset the control output when a boolean input switches to `true`. This controller is in the standard form

$$y(t) = k \left(e(t) + \frac{1}{T_i} \int e(s) ds + T_d \frac{de(t)}{dt} \right), \quad (2)$$

where we omitted for simplicity features of the implemented controller such as anti-windup, and where $y(t)$ is the control signal, $e(t) = u_s(t) - u_m(t)$ is the control error, with $u_s(t)$ being the set point and $u_m(t)$ being the measured quantity, k is the gain, T_i is the time constant of the integral term and T_d is the time constant of the derivative term. Note that the units of k are the inverse of the units of the control error, while the units of T_i and T_d are seconds.

As the units of flow rates and pressure can vary between orders of magnitude, for example depending on whether cfm , m^3/s or m^3/h are used for flow measurements, we decided to normalize the control error as follows: For temperatures, no normalization is used, and the units of k are $1/\text{Kelvin}$. No normalization is used because 1 Kelvin is 1.8 Fahrenheit , and hence these are of the same order of magnitude. For air flow rate control, the design flow rate is used to normalize the control error, and hence k is unitless. This also allows for using the same control gain for flows of different magnitudes, for example for a VAV box of a large and a small room, provided the rooms have similar transient response. For pressure control, the pressure difference is used to normalize the control error, and hence k is unitless.

Guideline 36 is specific as to where a P or a PI controller should be used. We used these recommendations as the default control configuration. However, all controllers can be configured as P, PI or PID controller. This allows for example to temporarily configure a PI controller as a P controller during the tuning process.

As Guideline 36 is written to convey the control intent rather than the actual implementation, it does not discuss how to avoid chattering of control due to sensor noise or numerical integration error. Therefore, for the part of the control sequences that use continuous time semantics, we added either hysteresis blocks or timers wherever the control or measurement signal is used as an input to a switch.²

5 Export of Control Sequences

We are currently developing a parser that exports CDL-conformant control sequences for the following use cases:

1. For human-readable documentation, the parser converts the sequences to html, similar to how Modelica tools generate html documentation.

²During the initial testing of the sequences, we indeed observed chattering and non-convergence of the solver as we missed a few of these switches.

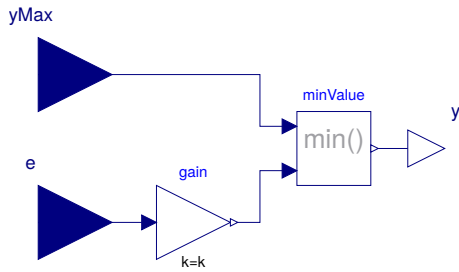


Figure 5. Graphical rendering of a the composite control block shown in Appendix A.

2. For further translation to control product lines, the parser converts the sequences to two JSON formats: One is an intermediate format that is close to the abstract syntax of Modelica, the other is generated by simplifying the former for easier processing by downstream applications. The latter representation is also used to generate html documentation of the control sequence.

The parser is currently being developed at <https://github.com/lbl-srg/modelica-json>. As an illustrative example, consider the composite block shown graphically in Figure 5 and textually using the CDL in Appendix A. The parser can export this specification to the JSON format shown in Appendix B.

The parser is implemented using ANTLR (ANother Tool for Language Recognition, <http://www.antlr.org/>) which converts CDL to a JSON format, which then is further simplified using JavaScript. For the html output, we use the mustache templating engine (<https://github.com/janl/mustache.js>).

6 Case Study

To test the suitability of CDL for simulation, we conducted closed loop simulations of multizone VAV sequences coupled to a whole building energy model. We will now summarize the experiment, and refer the reader for a more detailed description to Wetter et al. (2018) and <http://obc.lbl.gov/>. For the simulations, we used a model of one floor of the new construction medium office building for Chicago, IL, as described in the set of DOE Commercial Building Benchmarks (Deru et al., 2011). For all simulations, we used the same building and the same variable air volume flow system, but with two different control sequences. One sequence is based on the above described ASHRAE Guideline 36, whereas the other is the control sequence VAV 2A2-21232 of the Sequences of Operation for Common HVAC Systems (ASHRAE, 2006). All models are available in the Modelica Buildings library (Wetter et al., 2014), version 5.0.0, in the package `Buildings.Examples.VAVReheat`.

It turns out that changing the control sequence from VAV 2A2-21232 to the one published in Guideline 36 saves around 30% annual site HVAC energy under com-

parable thermal comfort. These are significant savings that can be achieved through software only, without the need for additional hardware or equipment. Moreover, the magnitude of these savings also questions how controls are typically represented in building energy simulation programs. Building energy simulation programs typically use idealized control sequences. These programs may then be used to compare the energy performance of different HVAC systems, such as a VAV system versus a radiant cooling system. However, such differences frequently are also in the order of 30%. Thus, to compare the energy performance of HVAC systems, control sequences must be represented adequately in the simulation, and the authors question the validity of the control idealizations that are commonly used in building energy simulation. If the variability due to controls is in the order of 30%, one cannot discern what apparent savings can be attributed to the change in HVAC system. Moreover, a process is needed that ensures that the control sequences will be implemented correctly and thus savings identified during design are realized during operation.

7 Conclusion

With the implementation of the Guideline 36 sequences and the case study, we have shown that our subset of the Modelica language that we identified for CDL suffices to implement control sequences for simulation. Ongoing work attempts to put in place a translator to a commercially available building automation system to see if unexpected issues arise that may require changes to CDL.

Our case study indicated that annual HVAC energy use can be reduced by 30% simply through the use of more sophisticated conventional control sequences. These sequences are however more complicated to specify and implement, and therefore we believe that for their proper use in design and actual operation of buildings, a process that allows their use in design, their (semi-automatic) translation to control product lines, and their verification relative to design specification is essential.

A key language issue that we selected to not support in the first version of CDL are state machines, for example as implemented in the `Modelica.StateGraph` package of the Modelica Standard Library 3.2 or as described in the Chapter 17 in the Modelica Language Definition (Modelica Association, 2012). The use of state machines would have made the implementation of control sequences considerably easier for blocks whose output is computed using various time delays, interlocks and modes of operation, which are frequently used in Guideline 36. As state machines are not universally supported in building automation systems, and as there are various flavors of state machines, we decided to currently not support them.

Selecting a subset of Modelica, in particular not supporting replaceable classes and multiple inheritance, considerably simplified the development of a translator from CDL to JSON. We believe that this also makes it easier for

control providers to support CDL.

At present, CDL supports conventional control sequences. In the future, CDL will also need to support control sequences that use Model Predictive Control or other advanced mathematical methods. How to provide blocks that can interface with such methods, or how to add vendor-specific packages that provide such advanced methods that are typically proprietary will be subject of future work.

8 Acknowledgment

This research was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy, under Contract No. DE-AC02-05CH11231, and the California Energy Commission's Electric Program Investment Charge (EPIC) Program.

References

- ALC, 2018. CtrlSpecBuilder, 2018. URL <https://www.ctrlspecbuilder.com>.
- ASHRAE. ANSI/ASHRAE Standard 135-2004, BACnet, a data communication protocol for building automation and control networks, 2004. ISSN 1041-2336.
- ASHRAE. *Sequences of Operation for Common HVAC Systems*. ASHRAE, Atlanta, GA, 2006.
- ASHRAE, 2016. *ASHRAE Guideline 36P, High Performance Sequences of Operation for HVAC systems, First Public Review Draft*. ASHRAE, June 2016. URL <http://gpc36.savemyenergy.com/public-files>.
- Marco Bonvini and Alberto Leva. A modelica library for industrial control systems. In *Proc. of the 9-th Int. Modelica Conf.*, pages 477–484, Munich, Germany, September 2012. Modelica Association. doi:DOI:10.3384/ecp12076477.
- Contemporary Controls, 2017. *Sedona Open Control – Reference Manual*. Contemporary Controls, September 2017. URL <https://www.ccontrols.com/pdf/RM-SEDONA00.pdf>.
- Michael Deru, Kristin Field, Daniel Studer, Kyle Benne, Brent Griffith, Paul Torcellini, Bing Liu, Mark Halverson, Dave Winiarski, Michael Rosenberg, Mehry Yazdani, Joe Huang, and Drury Crawley. U.S. Department of Energy commercial reference building models of the national building stock. Technical Report NREL/TP-5500-46861, National Renewables Energy Laboratory, Golden, CO, February 2011.
- Nicholas E.P. Fernandez, Srinivas Katipamula, Weimin Wang, YuLong Xie, Mingjie Zhao, and Charles D. Corbin. Impacts of commercial building controls on energy savings and peak load reduction. Technical Report 25985, PNNL, 5 2017.
- A. Husaunndee, R. Lahrech, H. Vaezi-Nejad, and J.C. Visier. Simbad: A simulation toolbox for the design and test of HVAC control systems. In Jean Jacques Roux and Monika Woloszyn, editors, *Proc. of the 5-th IBPSA Conf.*, pages 269–276, 1997. URL www.ibpsa.org/proceedings/bs1997/bs97_p022.pdf.
- Modelica Association, 2012. *Modelica – A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification, Version 3.3*. Modelica Association, May 2012. URL <https://www.modelica.org/documents/ModelicaSpec33.pdf>.
- Georg Ferdinand Schneider, Georg Ambrosius Peßler, and Simone Steiger. Modelling and simulation of standardised control functions from building automation. In *Proc. of the 12-th Int. Modelica Conf.*, pages 209–218, Prague, Czech Republic, may 2017. Modelica Association. doi:DOI:10.3384/ecp17132209.
- Siemens, 2000. *APOGEE Powers Process Control Language (PPCL) User's Manual*. Siemens Building Technologies, October 2000. URL https://www.quia.com/files/quia/users/hpiracer/AIRC65/PPCL_Users_Manual.
- George Thomas. *Creating an Open Controller with Sedona Framework™*. Contemporary Controls, February 2016. URL <https://sedona-alliance.org/pdf/WPSEDONAAA0.pdf>.
- Michael Wetter, Wangda Zuo, Thierry S. Nouidui, and Xiufeng Pang. Modelica Buildings library. *Journal of Building Performance Simulation*, 7(4):253–270, 2014. doi:DOI:10.1080/19401493.2013.765506.
- Michael Wetter, Jianjun Hu, Milica Grahovac, Brent Eubanks, and Philip Haves. OpenBuildingControl: Modeling feedback control as a step towards formal design, specification, deployment and verification of building control sequences. In *To appear in: 2018 Building Performance Modeling Conference and SimBuild*, September 2018.
- Y. Yang, A. Pinto, A. Sangiovanni-Vincentelli, and Q. Zhu. A design flow for building automation and control systems. In *2010 31st IEEE Real-Time Systems Symposium*, pages 105–116, November 2010. doi:10.1109/RTSS.2010.26.

Appendix A

The following statement, when saved as `CustomPWithLimiter.mo`, is the declaration of the composite block shown in Figure 3

```
block CustomPWithLimiter
  "Custom implementation of a P controller with variable output limiter"
  parameter Real k "Constant gain";
  CDL.Interfaces.RealInput yMax "Maximum value of output signal"
    annotation (Placement(transformation(extent={{-140,20},{-100,60}})));
  CDL.Interfaces.RealInput e "Control error"
    annotation (Placement(transformation(extent={{-140,-60},{-100,-20}})));
  CDL.Interfaces.RealOutput y "Control signal"
    annotation (Placement(transformation(extent={{100,-10},{120,10}})));
  CDL.Continuous.Gain gain(final k=k) "Constant gain"
    annotation (Placement(transformation(extent={{-60,-50},{-40,-30}})));
  CDL.Continuous.Min minValue "Outputs the minimum of its inputs"
    annotation (Placement(transformation(extent={{20,-10},{40,10}})));
equation
  connect(yMax, minValue.u1) annotation (
    Line(points={{-120,40},{-120,40},{-20,40},{-20, 6},{18,6}}, color={0,0,127}));
  connect(e, gain.u) annotation (
    Line(points={{-120,-40},{-92,-40},{-62,-40}}, color={0,0,127}));
  connect(gain.y, minValue.u2) annotation (
    Line(points={{-39,-40},{-20,-40},{-20,-6},{18,-6}}, color={0,0,127}));
  connect(minValue.y, y) annotation (
    Line(points={{41,0},{110,0}}, color={0,0,127}));
  annotation (Documentation(info="<html>
<p>
Block that outputs <code>y = min(yMax, k*e)</code>,
where
<code>yMax</code> and <code>e</code> are real-valued input signals and
<code>k</code> is a parameter.
</p>
</html>"));
end CustomPWithLimiter;
```

Appendix B

The JSON representation of the composite control block shown in Figure 5 is as follows, where we omitted the graphical annotations to keep the listing short.

```
[
  {
    "modelicaFile": "CustomPWithLimiter.mo",
    "topClassName": "CustomPWithLimiter",
    "comment": "Custom implementation of a P controller with variable output limiter",
    "public": {
      "parameters": [
        {
          "className": "Real",
          "name": "k",
          "comment": "Constant gain",
          "annotation": {
            "dialog": {
              "tab": "General",
              "group": "Parameters"
            }
          }
        }
      ]
    },
    "models": [
      {
        "className": "CDL.Interfaces.RealInput",
```

```

    "name": "yMax",
    "comment": "Maximum value of output signal"
  },
  {
    "className": "CDL.Interfaces.RealInput",
    "name": "e",
    "comment": "Control error"
  },
  {
    "className": "CDL.Interfaces.RealOutput",
    "name": "y",
    "comment": "Control signal"
  },
  {
    "className": "CDL.Continuous.Gain",
    "name": "gain",
    "comment": "Constant gain",
    "modifications": [
      {
        "name": "k",
        "value": "k",
        "isFinal": true
      }
    ]
  },
  {
    "className": "CDL.Continuous.Min",
    "name": "minValue",
    "comment": "Outputs the minimum of its inputs"
  }
]
},
"info": "<html>[omitted for brevity]</html>",
"connections": [
  [
    { "instance": "yMax" },
    { "instance": "minValue", "connector": "u1" }
  ],
  [
    { "instance": "e" },
    { "instance": "gain", "connector": "u" }
  ],
  [
    { "instance": "gain", "connector": "y" },
    { "instance": "minValue", "connector": "u2" }
  ],
  [
    { "instance": "minValue", "connector": "y" },
    { "instance": "y" }
  ]
]
}
]

```

Molten Salt–Fueled Nuclear Reactor Model for Licensing and Safeguards Investigations¹

M. Scott Greenwood¹

¹Oak Ridge National Laboratory, Oak Ridge, TN, USA greenwoodms@ornl.gov

Abstract

Fluid-fueled nuclear reactors, particularly molten salt reactors (MSRs), have recently gained significant interest. As with all reactors, modeling and simulation are key factors for advanced reactor design and licensing and will be required for the deployment of MSRs. However, there are significant gaps between simulation capabilities and system behavior for MSRs. This paper presents the system model of an MSR that is based on the Molten Salt Demonstration Reactor. The model includes important physics specific to MSRs, such as fission product and tritium transport and reactivity feedback.

Keywords: molten salt reactors, salt-fueled, nuclear

1 Introduction

The past few years have seen a significant increase in the interest of advanced fluid-fueled reactor systems, specifically molten salt reactors (MSRs). A *fluid-fueled reactor* is any reactor in which the fissile material is carried by the primary coolant throughout the primary flow circuit. Reactors such as MSRs could represent a revolutionary shift in the way nuclear power is implemented, and as a broad class of reactors, they have the potential to directly fulfill many US energy policy objectives.

Modeling and simulation are key for advanced reactor design and licensing. There are several modeling and simulation capabilities that can be used to investigate various aspects of nuclear reactors, including the Consortium for Advanced Simulation of Light Water Reactors (CASL) and the Nuclear Energy Advanced Modeling and Simulation Program (NEAMS). However, these have not been approved as licensing tools. Many of the capabilities from these and other programs are being adapted, or new ones are being created (Touran et al., 2017), to address the needs of advanced nuclear reactors and to ultimately generate tools that can be used for design and licensing of advanced reactors. Even so, significant technological

gaps are preventing rapid development of these tools, such as lack of data or difficulties in adapting legacy code intended for alternative applications such as LWR technologies.

Development of a low-fidelity, system-level model was identified as a straightforward path to identifying needs and gaps in data and simulation capabilities. For example, a system level model would allow for sensitivity analysis of dominating parameters that require additional research (e.g., heat and mass transfer coefficients) and for the exploration of safeguards and nuclear material accountancy that is an active area of research for MSRs. The identification of the various needs and gaps will inform required modifications to existing capabilities, help direct experimental data generation, and assist in requirement generation for modern high-fidelity code generation. This paper describes a system-level model of a thermal fluoride salt-fueled MSR that was created using the Modelica-based TRANSFORM tool developed by the Oak Ridge National Laboratory (ORNL) (Greenwood, 2017a). Additional details can be found in the following reference (Greenwood et al., 2018).

1.1 TRANSFORM

The TRANSient Simulation Framework of Reconfigurable Models (TRANSFORM) is a Modelica-based library developed at ORNL (Greenwood, 2017b; R. Hale et al., 2015). The tool's primary purpose is to provide a common simulation environment and baseline modeling resources to facilitate rapid development of dynamic advanced reactor models. Critical elements of this effort include (1) definition of standardized, common interfaces between models and components, (2) development of a library of baseline component modules to be assembled into full plant models using available geometry, design, and thermal-hydraulic data, (3) definition of modeling conventions for model and component development, and (4) establishment of user interfaces and support tools to facilitate simulation development and analysis (R. Hale et al., 2015; R. E.

¹Notice: This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (<http://energy.gov/downloads/doe-public-access-plan>).

Hale, Cetiner, et al., 2015; R. E. Hale, Fugate, et al., 2015). The TRANSFORM library has been successfully used for a variety of projects, including investigations into the performance of nuclear hybrid energy systems (Greenwood, 2017c; Greenwood, Cetiner, et al., 2017; Greenwood, Fugate, et al., 2017; Rabiti et al., 2017) and tritium transport (Rader et al., 2018).

1.2 Trace Substances

The Modelica language allows for a class called a *connector*, which allows a set of variables to always be defined between the connection of two models (e.g., in a fluid system: pressure, mass flow rate, specific enthalpy of the fluid, mass fraction of each species in the fluid, and trace substance mass weighted fraction). This fluid connector is implemented in the Modelica Standard Library and is adopted by TRANSFORM throughout its thermal-hydraulic library. A key feature of this variation of the fluid connector is the inclusion of a variable that accounts for trace substances. This work employs the trace substance variable to track the behavior of fission products, including delayed neutron precursors, principal contributors of reactivity feedback (e.g., xenon) and decay heat, and tritium.

A trace substance is assumed to be present in such small quantities that it has an insignificant impact on the mass of the system. Trace substances are tracked as unspecified mass-weighted fractions of the primary flow, so they flow as a homogenous part of the primary fluid, but they do not participate in the normal mass balance of the primary fluid. The absolute units of the traced substances are user definable based on the application. For example, if the primary fluid flows at 1 kg/s and a trace substance's (C) mass-weighted fraction is 100 atoms of C/kg fluid, then the primary fluid mass

balance assumes that there is 1 kg/s of primary fluid, and the trace substance mass has its own mass balance tracking the 100 atoms/s flowing through the system. This method allows for mapping of small quantities of substances in traditional thermal-hydraulic processes as a first-order approximation, obviating the need to create complex media properties, pressure loss functions, etc. The behavior of the primary fluid is driven by its own mass, energy, and momentum balances.

2 Model Criteria

MSR development requires integrated performance models to understand the interaction and feedbacks between systems. As this model was intended to inform the development of salt-fueled reactors, necessary requirements of the model were identified from licensing and safeguards considerations. In broad terms, licensing and safeguards may be defined as follows:

- *Licensing*: The process by which the US Nuclear Regulatory Commission (NRC) ensures the protection of public health and safety, the common defense and security, and the environment (NRC 2010). Of primary importance are the types of radioactive sources and the pathways of exposure of those sources to site personnel and the public.
- *Safeguards*: “the timely detection of diversion of significant quantities of nuclear material from peaceful nuclear activities to the manufacture of nuclear weapons or of other nuclear explosive devices or for purposes unknown, and deterrence of such diversion by the risk of early detection” (Paragraph 28, INFCIRC/153) (IAEA, 1972).

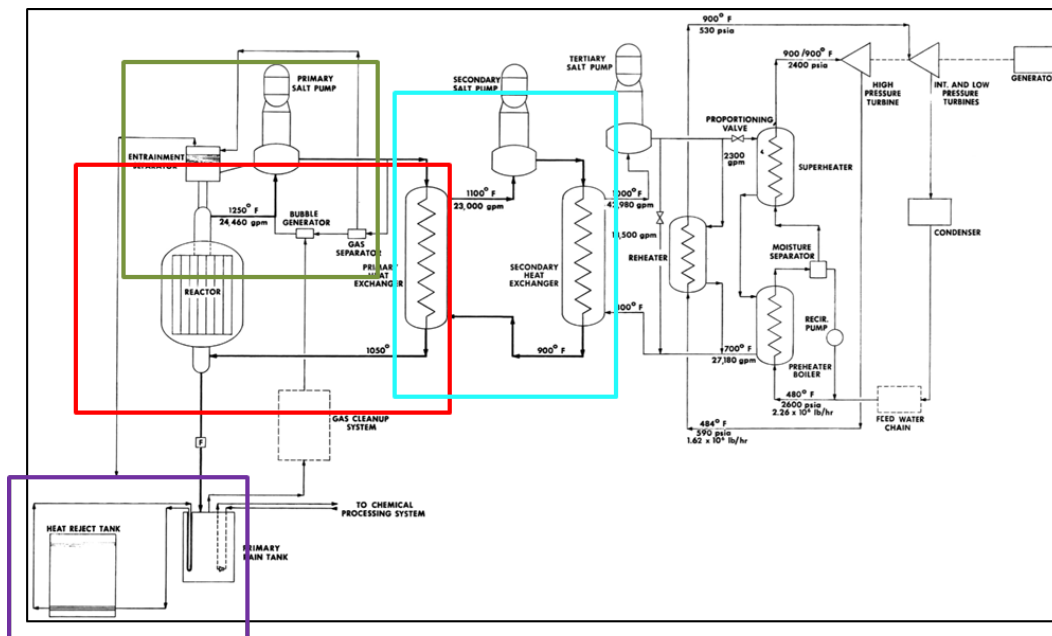


Figure 1. Simplified flowsheet of the salt-fueled thermal MSDR. Boxes represent corresponding systems between the MSDR flowsheet and the system model.

In addition to licensing and safeguards, available data, historical experience, and current proposed MSR designs were also evaluated to guide the design criteria of the system model. Historical operational experience is limited to two small salt-fueled, thermal spectrum demonstration reactors, both of which were operated at ORNL (Rosenthal 2009). The Aircraft Reactor Experiment (ARE) was operated in 1954, and the Molten Salt Reactor Experiment (MSRE) was operated from 1965–1969. Numerous design documents, including balance-of-plant and auxiliary systems documents, were generated under the Molten Salt Breeder Reactor (MSBR) Program in the 1960s and 1970s. Modern MSR designs will need to understand the source term behavior in their systems for licensing and safeguards considerations, as well as performance of auxiliary systems such as off-gas and decay heat removal systems. The criteria required of the dynamic models are enumerated below.

1. Inclusion of delayed neutron precursor models that account for delayed neutron precursor production, transport, and decay throughout the primary fueled reactor loop (i.e., reactor core to primary heat exchanger and back) and auxiliary systems
2. Radionuclide inventory accounting, including source term production and holdup and release mechanism models
3. Thermal-hydraulic analyses of sufficient fidelity to capture flow and power dynamics in salt-fueled concepts
4. Time-, temperature-, flux-, and flow-dependent materials and salt interaction data, and models to predict corrosion, erosion, and irradiation effects

5. Modeled concepts that rely on existing data where possible to minimize development time while remaining relatively generic and applicable to modern MSR designs

3 Model Development

3.1 Reference Design

In accordance with the identified model criteria, the Molten Salt Demonstration Reactor (MSDR) (Bettis et al., 1972) provides the base design concept for the fluoride salt-fueled reactor dynamic model (Figure 1), with the exception that a U/Pu fuel salt is used rather than the Th-fueled salt of the original concept. This concept has a nominal thermal output of 750 MWt. The purpose of the MSDR was to demonstrate the molten-salt reactor concept on a semi-commercial scale while minimizing development of basic technology beyond that already demonstrated by the MSRE.

An advantage of basing the reactor concept on an existing design is attributable to the detailed design document developed by researchers who were intimately familiar with the MSRE technology. The MSDR also leverages the work of the MSBR (Robertson, 1971) for information on off-gas, chemistry, materials, neutron physics, fuel reprocessing, etc., an effort which was also carefully documented. This model provides information that directly applies to the development of commercial reactors, minimizing development requirements and complication of systems. These elements help meet the near-term deployment targets of modern vendors.

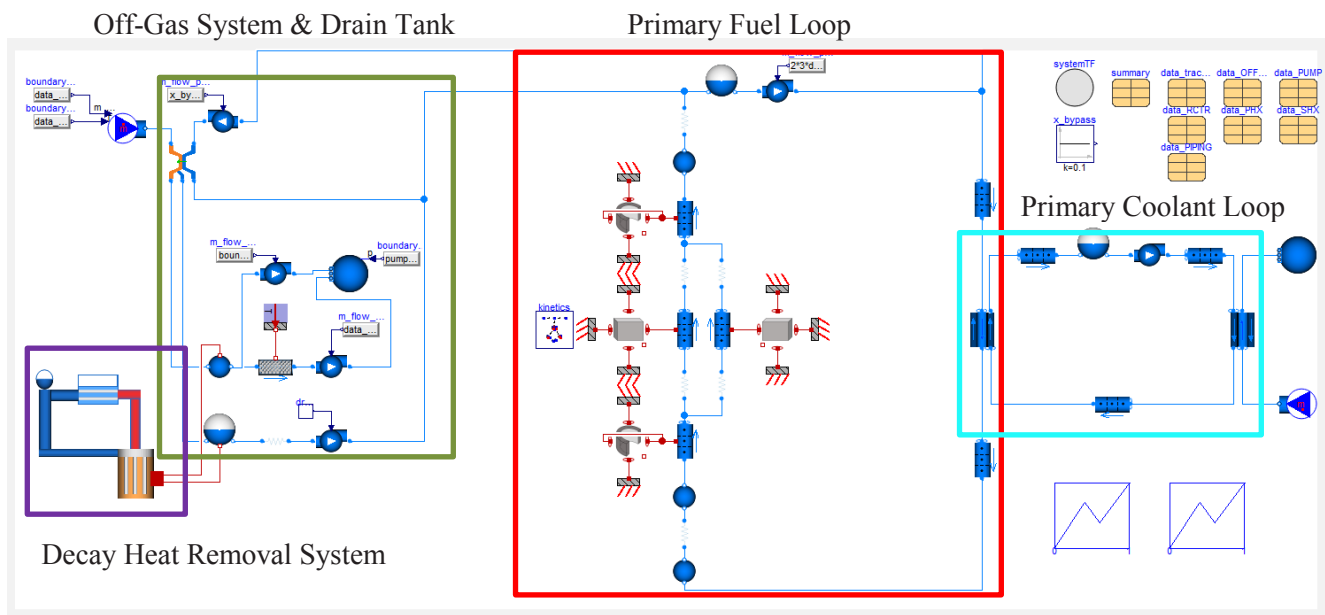


Figure 2. System model of a fluoride salt-fueled thermal reactor based on the MSDR. Boxes represent corresponding systems between the MSDR flowsheet and the system model.

3.2 System Model

The system model (Figure 2) was developed based on the available MSDR literature and the identified criteria. Specific subsystems and other important phenomena are discussed in more detail below. Other critical systems that impact the reactor behavior such as the balance of plant (BOP) will be modeled in the future.

3.2.1 Primary Fuel and Coolant Loop

The primary fuel loop (PFL) is defined as the primary circuit of fuel salt, including the reactor, the primary fuel pump, piping to and from the primary heat exchanger, and the primary heat exchanger (fuel side).

The principal component of the PFL is the reactor (Figure 3). The reactor consists of an inlet and outlet plenum, two axial reflectors, a radial reflector, and the core. Since the surface area and volume of graphite and fuel salt are important not only for heat transfer considerations but also for interaction with trace substances, a concerted effort was made to preserve reactor geometries in the model.

The identical inlet and outlet plenums are ideally mixed volumes. The identical inlet and outlet graphite reflectors are made from radial rings of graphite, with each ring consisting of several smaller sections of

graphite. The graphite is modeled as a 2D radial (r - z) conduction model with a specified number of parallel graphite blocks. Heat and mass transfer are modeled on the inner and outer radial surfaces and neglected on the top, bottom, and edge. The fluid subchannel is represented by a 1D discretized homogeneous fluid, with geometry specified by the total cross section area and the wetted perimeter of the reflector.

The radial reflector consist of stacked rectangular slabs and is therefore modeled with a 2D slab (x - z) conduction model. The slab is assumed to have an adiabatic centerline which permits modeling only half of the slab. The entire slab can be modeled by increasing the number of parallel characteristic solids by a factor of two. Heat and mass transfer are neglected on the top, bottom, and small edge of the block.

The core region graphite (Figure 4) is also modeled with a 2D slab (x - z) conduction model with appropriate dimensions. The fluid channel was determined by the cross-sectional flow area and the wetted perimeter, in association with their respective graphite.

The pipes to and from the primary fuel heat exchanger dimensions are approximated based on rough estimations from drawings, as their dimensions were not specified in available documentation. Likewise, the pump and pump bowl were never fully defined, so the dimensions of the pump bowl were assumed to be

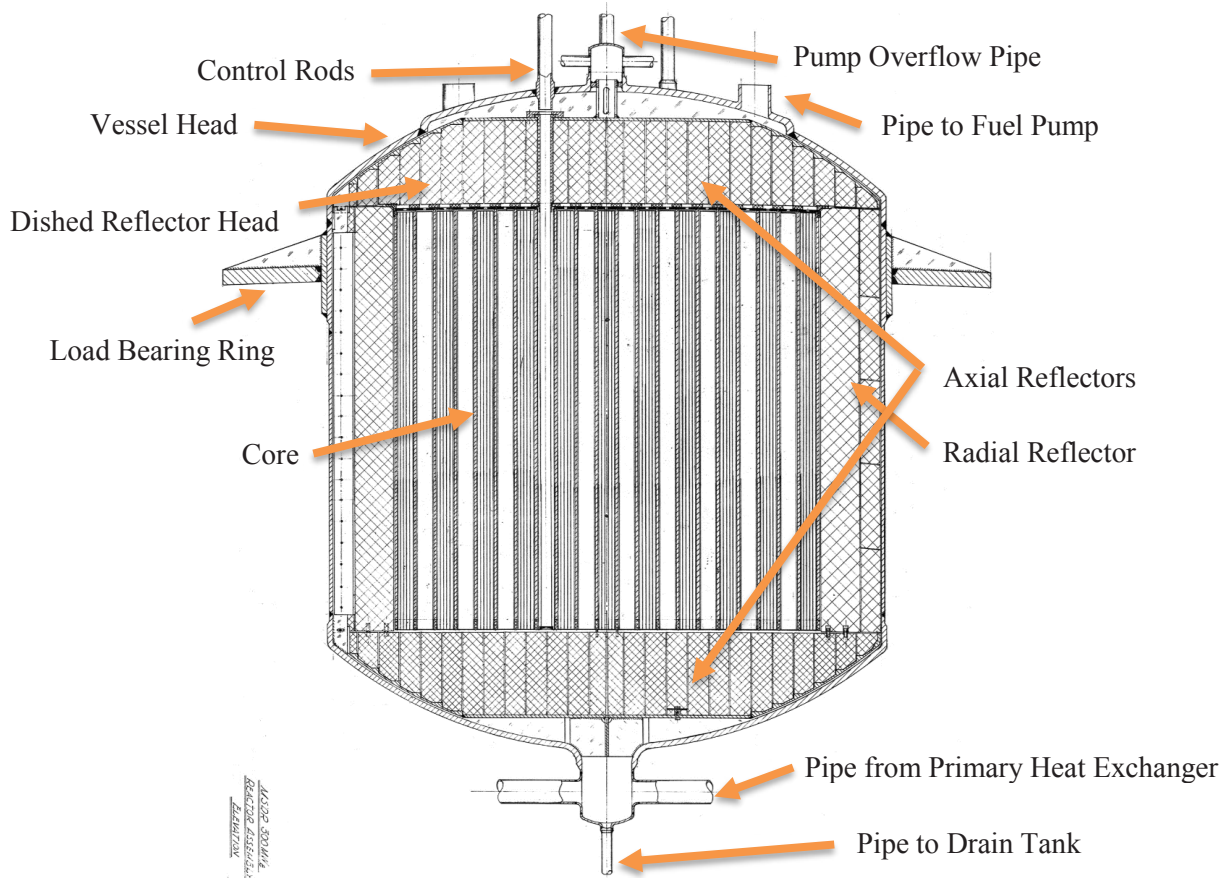


Figure 3. MSDR reactor vessel geometry (Bettis, Alexander, and Watts 1972, Fig. 2, ORNL DWG 72-2829).

similar to those in the MSBR. The off-gas system interfaces with the primary loop at the pump bowl inlet and the pump outlet by cycling a fraction of the overall flow through a separator to strip fission product gas products, and then returns the remaining fluid and fission products to the pump bowl. A small amount of fuel salt also leaves the system and travels to the drain tank, where it is pumped back to the pump bowl. The amount of salt sent to the drain tank and back to the pump bowl depends on the flow rate of carrier gas for the off-gas system and the level controls of the drain tank pumping system. Additional information on the off-gas and drain tank systems is detailed in Section 3.2.4.

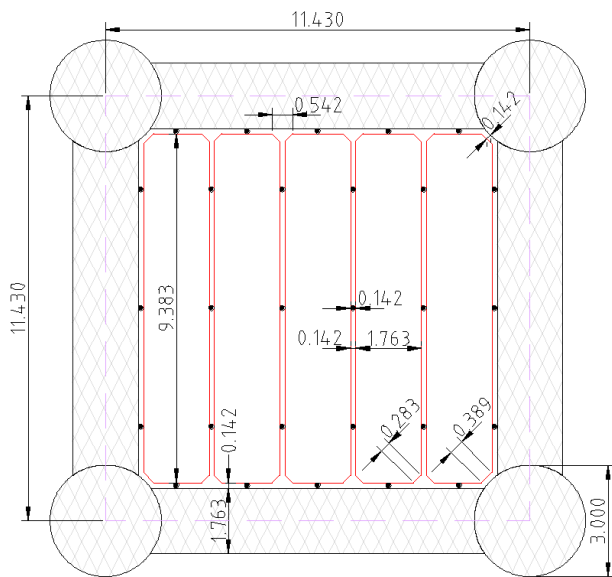


Figure 4. Reproduction of core cell (Bettis, Alexander, and Watts 1972, Fig. 7; ORNL DWG 72-2830). Dimensions in inches; the dashed magenta line indicates a unit cell.

The primary fuel heat exchanger is a simple shell-and-tube heat exchanger (Bettis et al., 1972). The primary coolant loop (PCL) is defined as the primary circuit of coolant salt, which includes the primary heat exchanger (coolant side), the coolant pump, the secondary heat exchanger (coolant side), and associated piping.

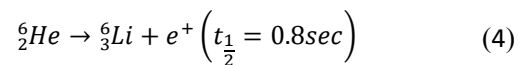
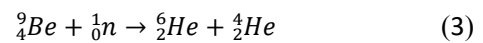
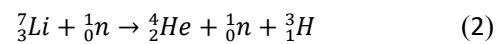
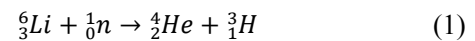
3.2.2 Reactor Kinetics & Fission Product Transport

The reactor kinetics implementation is a critical component of the system model, as it determines the reactor power output and establishes the behavior of the source terms (generation, reactivity feedback, etc.) throughout the system. The current implementation is based on a modified form of the point reactor kinetics (Greenwood and Betzler, In Review). This model captures the creation, decay, and transport of fission products and the reactivity feedback of neutron absorbers such as xenon, and it includes decay heat in terms of the near- and far-field energy deposition associated with fission product.

Fission product substances such as neutron precursor groups are treated as trace substances transported at the same rate as the primary carrier fluid, as described in the Introduction above. Fission product concentrations and their associated decay are tracked for all fluid volumes throughout the entire system.

3.2.3 Tritium Transport

Tritium (3_1H) is generated in significant quantities in MSR. Its source is primarily from interaction of lithium-6 (6_3Li) in the carrier salt with neutrons in the reactor vessel. For example, Eqs. (1–4) summarize the major production pathways of tritium in a FLiBe-based system (Stempien, 2015).



To account for this production, an additional source term for tritium is included in the modified reactor kinetics model based on the composition of the fluid.

Tritium differs from other fission products due to (1) its generation by interaction of salt with neutrons, and (2) the manner in which it readily diffuses through piping at the elevated operating temperatures of MSRs, especially through the thin walls of the heat exchangers (Mays et al., 1977). Using mass transfer analogies to heat transfer, the tritium is permitted to flow from the primary fuel loop to the primary coolant loop and from the primary coolant loop to the balance of plant through the respective heat exchangers. Further details on the methodology can be found in (Rader et al., 2018). Other major components accountable for tritium leaving the primary fuel loop, or tritium management systems, may be incorporated in the future.

3.2.4 Auxiliary Systems

An actual MSR, like any industrial scale facility, will have many auxiliary systems. Three systems important to MSRs that are included in the system model are the off-gas system, the drain tank system, and the decay heat removal system. While these systems are important for understanding performance and source term behavior, they are not well defined in literature. Therefore, engineering judgment and simplifications were made for preliminary modeling purposes.

The off-gas system removes a specified set of fission products (i.e., gaseous products) from the primary fuel salt pump bypass line at a specified efficiency using a helium carrier gas (Figure 2). A portion of primary fuel salt is also carried from the primary fuel loop at a rate

dependent on the carrier gas flow rate. The separated fuel salt travels directly to the drain tank, where it is then pumped back to the pump bowl of the primary fuel loop. The rate of fuel salt return from the drain tank can be controlled using the control settings of the drain tank sump pump. The carrier gas with the separated fission products also travels to the drain tank. The characteristic hold-up time of the gas depends on the tank volume. From the drain tank, the gas is split at a specified ratio between a return line that runs directly back to the pump bowl and a charcoal adsorber bed. As the gas passes through the charcoal bed, substances decay, give off heat, and may become trapped. After exiting the charcoal bed, the carrier gas, along with any remaining substances that did not completely decay or that were otherwise filtered, are returned to the pump bowl.

The charcoal bed transports (Sun et al., 1994) the trace substances between volumes in the adsorber bed at a rate ($m\dot{C}$) which is a function of the inflow rate, the time spent in a volume (τ), the decay rate of the substance (λ), and any sources of each substance from the decay of other substances, as shown in Eq. (5):

$$m\dot{C}_{out} = m\dot{C}_{in}e^{-\lambda\tau} + \sum m\dot{C}_{decay} \quad (5)$$

The adsorber bed is heated by the decay of fission products. Like the drain tank heat removal system, the adsorber bed is cooled by a passive circulation loop. For simplicity, a fixed boundary temperature is set for the adsorber bed so that the cooling requirement can be easily monitored, as no design information is available for that system.

The drain tank is separated into two volumes (Figure 2), one for the gas and one for the fuel salt. The gas volume is determined by the liquid level of the fuel salt in the specified geometry, while the pressure of the fuel salt volume is set by the gas volume. Products decay and emit heat in each of these volumes and then continue through the process. The gas volume continues to the adsorber bed or goes directly to the pump bowl as previously discussed, while the fuel salt is pumped back to the pump bowl based on the control algorithm implemented. The preliminary control is a level monitor which switches its control setting based on minimum and maximum fuel salt levels. The drain tank is thermally connected with the decay heat removal system through double-walled thimbles, as described below.

The decay heat removal system (Figure 5) is a passive, buoyancy driven, NaK-filled circulation loop. The loop removes heat from the drain tank via double walled thimbles, which rely on radiation heat transfer between the pipes and convective heat transfer between the working fluids and the pipe walls. The hot fluid rejects to a water tank at a higher elevation via identical double-walled thimbles. The cold fluid recirculates back to the drain tank to be reheated. As this system is not well defined, a flow resistance is inserted into the loop so that the mass flow rate matches the design references. This resistance would be comprised of bends, orifices, and other pressure losses not already accounted for by the pressure drop correlations in the pipes. The water tank has a simple control system that maintains the outlet temperature at the design condition. The current model of the water tank includes a simple, ideally mixed volume that does not consider latent heat effects, so the

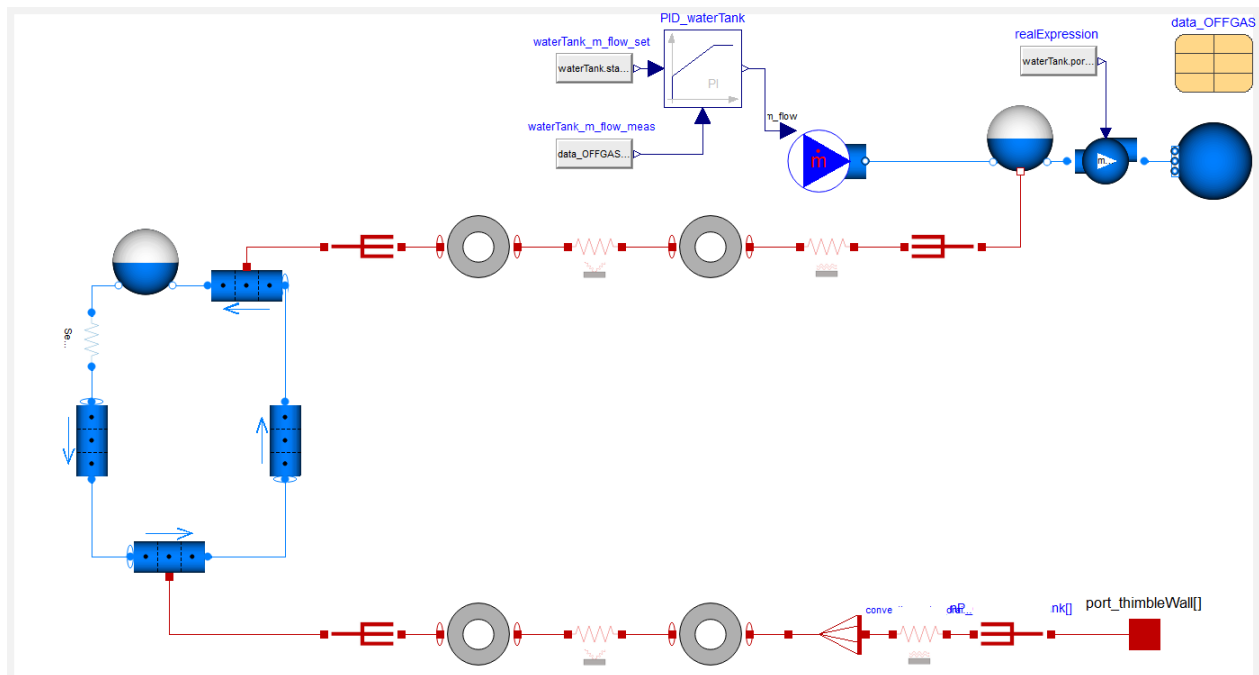


Figure 5. Model of the drain tank natural circulation decay heat removal system.

flow rate required to keep the tank at design conditions is overestimated.

4 Results

There are many various scenarios of potential interest to the regulator body and MSR community. A subset of accident and normal operation scenarios include the following:

- Loss of power
- Single and multiple pump failures
- A variety of reactivity insertion/removal events
- Tritium dose to the environment
- Fission product inventory
- Decay heat removal performance
- Load following ability
- Overall passive safety performance

This list can be quite extensive. Two scenarios are presented herein. The first scenario is a steady-state case with a focus on demonstrating fulfillment or progress on the model criterion. This steady state case also serves as the initial condition for the second case. The second scenario is an accident type scenario in which the primary fuel pump trips, after which the primary coolant pump trips. Under both scenarios, the temperature feedback for the point kinetics equations establishes the power level. At full power, this temperature feedback is assumed to be linear between the nominal inlet and outlet temperature. Exact dimensions/parameters are excluded, and results are normalized, as the cases are meant to demonstrate overall behaviors and capabilities. Simulations were run using Dymola 2018 FD01. Details of the simulation can be found in Table 1.

Table 1. Simulation Parameters

| <i>Parameter</i> | <i>Value</i> |
|------------------|--------------|
| Simulation Time | 172800 s |
| Real Time | 260 s |
| Solver | Esdirk45a |
| Tolerance | 0.0001 |
| Equations | 13290 |
| Scalar Equations | 58082 |

4.1 Case 1: Steady State Behavior

A steady-state condition was reached by simulating the model for 172,800 seconds (2 days). The extensive simulation time was due to permitting the reactor to start from zero concentration fission product and therefore required a sufficient length of time for longer lived produces, like xenon (half-life on the order of 9 hours) to build-up to steady-state concentrations.

Figure 6 demonstrates the expected skew of temperature feedback and power based on the assumed linear increase in reference temperature as specified in the case description. The power at the inlet of the core is near zero (though decay heat is still generated) as the fluid enters near the nominal inlet temperature. The power in the subsequent nodes increases due to the temperature feedback, although the profile shifts toward the core outlet due to movement of the precursor neutrons. The temperature difference between the measured and reference temperature decreases toward the outlet of the reactor, decreasing the temperature feedback and slightly lowering the power level of the reactor.

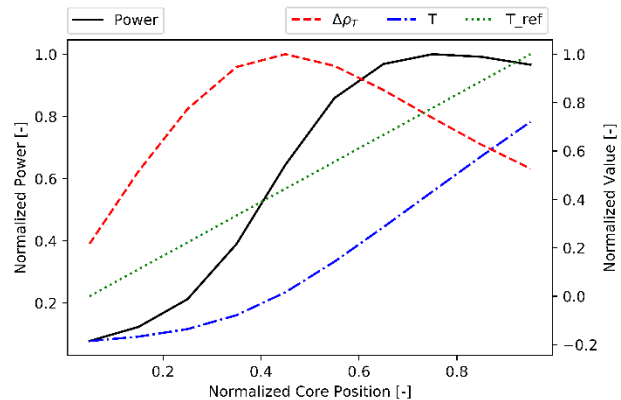


Figure 6. Normalized power and temperature reactivity feedback in the core.

Figure 7 presents the concentration of tritium, a selected neutron precursor group with a relatively short half-life, and xenon as a function of position in the reactor model. The tritium is generated in the core, dependent on the power profile, and then it diffuses through the PFL heat exchanger (HX) to the PCL. Little variation is seen elsewhere due to the long half-life of tritium (~12 years). Neutron precursors, the principle feedback mechanism for point kinetics, are typically separated into several groups. For discussion purposes, only one of the groups, which has a half-life on the order of seconds, is presented. The plot demonstrates the behavior of this group as its generation rate closely follows the power profile of the core and then decays to nearly zero between the core outlet and the PFL HX inlet. Like tritium, xenon's decay rate is slow compared to the loop transit time. However, removal of xenon to the off-gas and charcoal adsorber bed for decay hold-up can be seen in the figure at a position of approximately 10-m. The pump bowl is at this position, which has the associated separation process previously described.

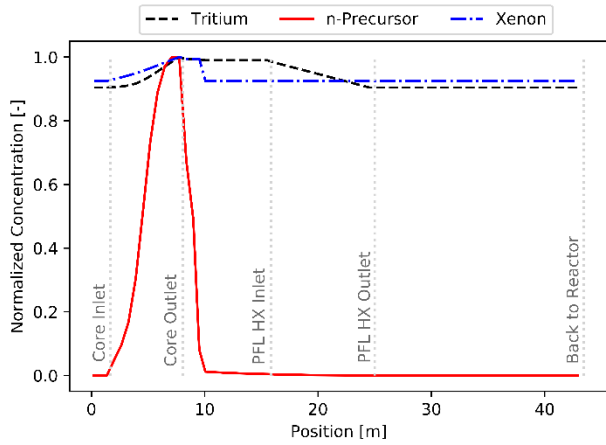


Figure 7. Normalized concentrations of Tritium, a neutron precursor group, and xenon as a function of position in the reactor model.

4.2 Case 2: Sequential Pump Trips

Following the two-day simulation, the reactor pumps for the PFL and PCL were ramped down by 95% from the full power operation flow rate over a period of 60 seconds. The PFL was tripped first, followed by the PCL pump five minutes later. This process can be seen in Figure 8.

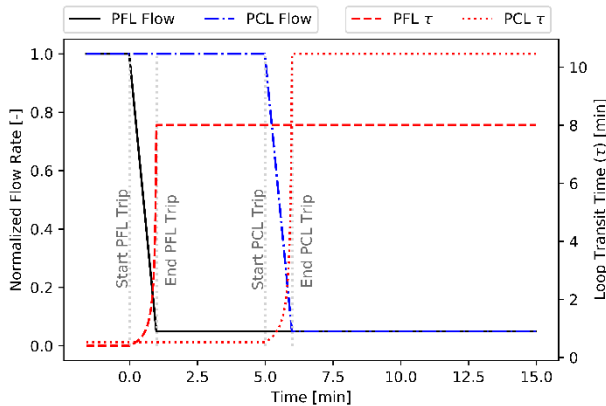


Figure 8. Pump trip event showing the sequential coast down of the PFL and PCL pumps and the associated impact on the respective loop transit times.

Figure 9 – Figure 12 illustrates the feedback on the system due to the drop in PFL flow rate. Upon PFL pump trip, the temperature within the core of the reactor heats up due to the decreased flow rate (Figure 9). As the temperature increases, the temperature feedback of the reactor drives the power down (Figure 10) as it attempts to correct the discrepancy between the reference and measured nodal temperatures. The oscillations in temperature, and therefore reactivity and power, is associated with the influx of colder-than-nominal fluid returning from the PFL HX. This fluid is cooled more than usual due to the continued operation of the PCL pump. For clarity Figure 11 and Figure 12 are both presented. These plots illustrate the time-

dependent behavior of the entire loop temperature. At the time of the PFL pump trip, the increase in core temperature and decrease in HX temperature are clearly seen. Once the PCL pump trips, the system shifts slightly back toward the nominal condition, as the temperature feedback once again compensates for the off-nominal temperature differences.

Although the change in a variety of behaviors in the PFL is noticeable in the simulation, in the decay heat removal system the heat removal response is very limited, as would be expected since moderate- to long-lived isotopes are the primary contributors of decay heat (Figure 10).

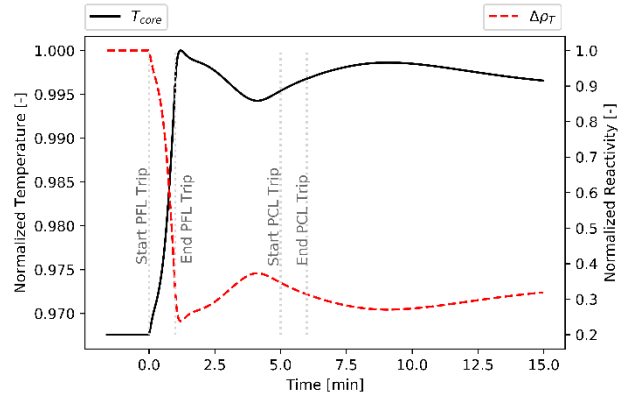


Figure 9. Normalized core temperature and thermal reactivity feedback as a function of time.

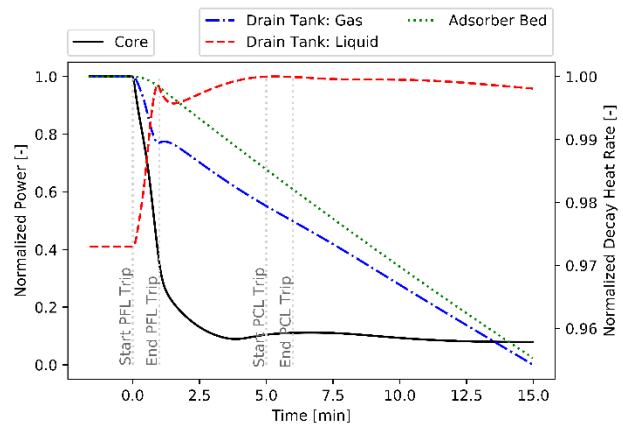


Figure 10. Normalized power generated in the core and removed in the decay heat rejection system.

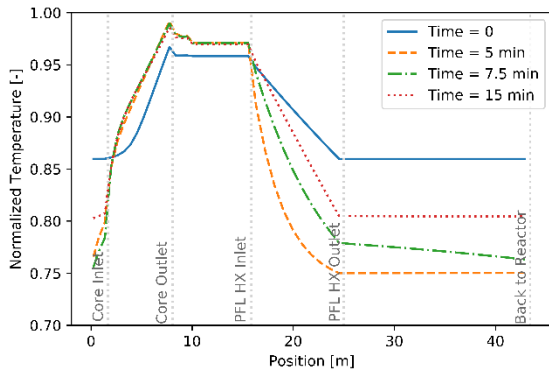


Figure 11. Normalized PFL temperature as a function of loop position for select times throughout the transient case.

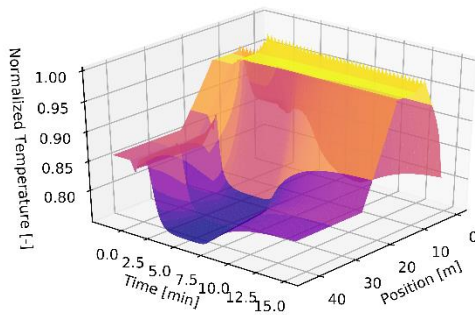


Figure 12. Time history of the evolving PFL temperature as a function of position in the loop.

The tritium release rate oscillates as a function of changes in fluid densities, generation rates, etc. Figure 13 demonstrates the response throughout the system. As expected, the tritium generation rate tracks the power of the reactor. As the initial PFL pump trips the release rate slowly rises, decreases, and rises again. This behavior is due to oscillations in the fluid density (change in tritium volumetric concentration) as temperature waves from the transient case propagate through the system. Once the PCL trips, temperature, and therefore densities and concentrations, return to conditions close to nominal. The difference in release rate of tritium at the beginning and end of simulation are approximately equal due to the amount of tritium being released being a small fraction of the actual amount of tritium in the system. Over longer periods of time, on the order of days, the release

rate decreases to match the generation rate.

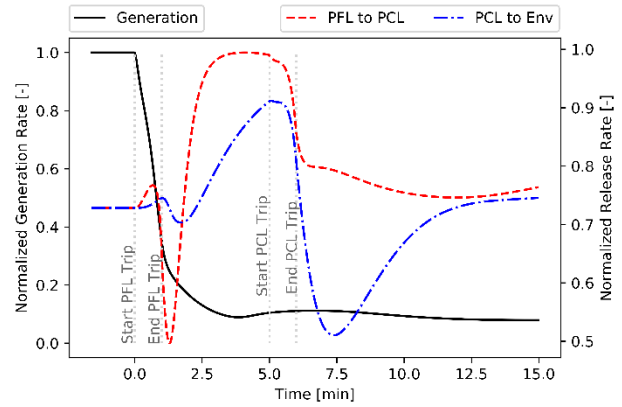


Figure 13. Tritium generation and release rate from the PFL to the PCL and the PCL to the environment (or BOP).

5 Summary

To advance the understanding of molten salt-fueled nuclear reactors, a low-fidelity system-level model has been generated in Modelica using the ORNL-developed TRANSFORM library. The simulation relies on the trace substance methodology introduced in the MSL and adopted in TRANSFORM to account for the transport of species, such as fission products, and their impact on the system, including reactivity feedback and decay heat generation. The paper presents a select set of data for steady state and pump-trip scenarios to demonstrate the capabilities and implemented physics of the model. This model will continue to be extended and tailored to specific examples and demonstrations to help identify needs and gaps in data and simulation capabilities in current and future nuclear reactor licensing and design tools. More generally, the presented model and fission product modeling approach demonstrates the ability to advance the understanding of the dynamics of complex fluid-fueled reactor systems, a critical part of licensing and safeguards analysis, for which few if any tools exist.

Acknowledgments

The author would like to acknowledge and thank the US Department of Energy for funding this work.

References

- Bettis, E. S., Alexander, L. G., and Watts, H. L., 1972. *Design Studies of a Molten-Salt Reactor Demonstration Plant* (No. ORNL-TM-3832). Retrieved from <http://moltsalt.org/references/static/downloads/pdf/ORNL-TM-3832.pdf>
- Greenwood, M. S., 2017a. *TRANSFORM-Library: A Modelica based library for modeling thermal hydraulic energy systems and other multi-physics systems*. Modelica, Retrieved from <https://github.com/ORNL-Modelica/TRANSFORM-Library>
- Greenwood, M. S., 2017b. *TRANSFORM - TRANSient Simulation Framework of Reconfigurable Models*.

- Modelica, Oak Ridge National Laboratory. DOI: 10.11578/dc.20171109.1
- Greenwood, M. S., 2017c. Component Development for Nuclear Hybrid Energy Systems (pp. 839–846). Presented at the 12th International Modelica Conference, Prague, Czech Republic. DOI: 10.3384/ecp17132839
- Greenwood, M. S., and Betzler, B. R., In Review. Modified Point Kinetic Model for Neutron Precursors and Fission Product Behavior for Fluid-Fueled Molten Salt Reactors. *Nuclear Science and Engineering*.
- Greenwood, M. S., Betzler, B. R., and Qualls, A. L., 2018. *Dynamic System Models for Informing Licensing and Safeguards Investigations of Molten Salt Reactors* (No. ORNL/TM-2018/876, 1456790). DOI: 10.2172/1456790
- Greenwood, M. S., Cetiner, S. M., Harrison, T. J., and Fugate, D. L., 2017. *A Templated Approach for Multi-Physics Modeling of Hybrid Energy Systems in Modelica* (No. ORNL/TM-2018/757). Oak Ridge National Lab. (ORNL), Oak Ridge, TN (United States).
- Greenwood, M. S., Fugate, D. W., and Cetiner, S. M., 2017. Control Systems for a Dynamic Multi-Physics Model of a Nuclear Hybrid Energy System. Presented at the 10th International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies, San Francisco, CA (United States).
- Hale, R. E., Cetiner, S. M., Fugate, D. L., Batteh, J. J., and Tiller, M. M., 2015. *Update on Small Modular Reactors Dynamic System Modeling Tool: Web Application* (No. ORNL/SPR--2015/17). Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States). DOI: 10.2172/1252137
- Hale, R. E., Fugate, D. L., Cetiner, S. M., and Qualls, A. L., 2015. *Update on ORNL Transform Tool: Simulating Multi-Module Advanced Reactor with End-to-End I&C* (No. ORNL/SPR-2015/257). Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States). DOI: 10.2172/1239756
- Hale, R., Fugate, D. L., Cetiner, S. M., Ball, S. J., Qualls, A. L., and Batteh, J. J., 2015. *Update on ORNL TRANSFORM Tool: Preliminary Architecture/Modules for High-Temperature Gas-Cooled Reactor Concepts and Update on ALMR Control* (No. ORNL/SPR--2015/367). Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States). Retrieved from <https://info.ornl.gov/sites/publications/Files/Pub56863.pdf>
- IAEA, 1972. *The Structure and Content of Agreements Between the Agency and States Required in Connection with the Treaty on the Non-Proliferation of Nuclear Weapons* (No. INFCIRC/153 (Corrected)). International Atomic Energy Agency. Retrieved from <https://www.iaea.org/publications/documents/infcircs/structure-and-content-agreements-between-agency-and-states-required-connection-treaty-non-proliferation-nuclear-weapons>
- Mays, G. T., Smith, A. N., and Engel, J. R., 1977. *Distribution and Behavior of Tritium in the Coolant-Salt Technology Facility* (No. ORNL/TM-5759). Oak Ridge National Lab. (ORNL), Oak Ridge, TN (United States).
- Rabiti, C., Epiney, A. S., Talbot, P., Kim, J. S., Guler Yigitoglu, A., Greenwood, M. S., Cetiner, S. M., et al., 2017. *Status Report on Modelling and Simulation Capabilities for Nuclear-Renewable Hybrid Energy Systems* (No. NL/EXT-17-42441). Idaho National Laboratory.
- Rader, J. D., Greenwood, M. S., and Humrickhouse, P. W., 2018. Verification of Modelica-Based Models with Analytical Solutions for Tritium Diffusion. *Nuclear Technology*, 1–8. DOI: 10.1080/00295450.2018.1431505
- Robertson, R. C., 1971. *Conceptual Design Study of a Single-Fluid Molten-Salt Breeder Reactor* (No. ORNL-4541). Oak Ridge National Laboratory. Retrieved from <https://www.osti.gov/scitech/servlets/purl/4030941>
- Stempien, J. D., 2015, May. *Tritium Transport, Corrosion, and Fuel Performance Modeling in the Fluoride Salt-Cooled High-Temperature Reactor (FHR)*. Cambridge, MA: Massachusetts Institute of Technology.
- Sun, C. L., Chen, J. C., Yu, Y. W., Ha, H. C., Lu, C. S., and Lee, T. Y., 1994. Dynamic adsorption properties of Kr and Xe isotopes in charcoals. *Journal of Radioanalytical and Nuclear Chemistry*, **181**(2), 291–299. DOI: 10.1007/BF02037635
- Touran, N. W., Gilleland, J., Malmgren, G. T., Whitmer, C., and Gates, W. H., 2017. Computational Tools for the Integrated Design of Advanced Nuclear Reactors. *Engineering*, **3**(4), 518–526. DOI: 10.1016/J.ENG.2017.04.016

Development and Implementation of a Flexible Model Architecture for Hybrid-Electric Aircraft

John Batteh¹ Jesse Gohl¹ Michael Sielemann² Peter Sundstrom³ Ivar Torstensson³
 Natesa MacRae⁴ Patrick Zdunich⁴

¹Modelon Inc., USA, {john.batteh, jesse.gohl}@modelon.com

²Modelon Deutschland GmbH, Germany, {michael.sielemann}@modelon.com

³Modelon AB, Sweden, {peter.sundstrom, ivar.torstensson}@modelon.com

⁴National Research Council Canada, Canada, {Natesa.MacRae, Patrick.Zdunich}@nrc-cnrc.gc.ca

Abstract

This paper describes the implementation of a flexible, modular, hybrid-electric aircraft modeling architecture for the development of a virtual and physical demonstrator system that will be used in the advancement of sustainable mobility systems by the National Research Council of Canada (NRC). The initial modeling architecture was established in Modelica based on the NASA X-57 electric flight demonstrator aircraft. A series of models were assembled from a high level aircraft system architecture to mimic the initial developmental path from the baseline conventional aircraft to the X-57 electric aircraft variant. The multi-physics component models describe the aircraft dynamics and performance, integrated with the relevant mechanical, electrical, and thermal dynamics of the electric aircraft power train. The proposed modular architecture allowed the simulation of three different aircraft configurations with different degrees of electrification, demonstrating its effectiveness and versatility in the design and development of hybrid-electric aircraft.

Keywords: aerospace, hybrid-electric aircraft, electrification, electric propulsion, thermal

1 Introduction

Triggered by the demand for reduced emissions / fuel burn, external noise, and maintenance costs, the aircraft industry is actively pursuing the concepts of ‘more electric’ and ‘fully electric aircraft’. Although the first manned electric aircraft flight took place in 1973 with the Militky MB-E1 (Taylor, 1974), a growing trend has emerged in the past decade with the increased electrification of the Boeing 787 Dreamliner (Boeing, 2018), the all-electric Airbus E-Fan (with recent plans for the collaborative development of the E-Fan X between Airbus, Rolls Royce and Siemens) (E-Fan/E-Fan X, 2017), the Pipistrel Alpha Electro (Pipistrel, 2018), and the ongoing development of the National Aeronautics and Space Administration (NASA) X-57 (NASA, 2018). Current advancements in the technologies that support this trend include innovations in the fields of power electronics, sensors, high density electric motors, energy storage, and power generation. In addition, advances in modeling and simulation software have led to the development of flexible, modular model architectures, which provide a uniquely efficient way to manage the challenges inherent in the modeling of advanced multi-domain systems.

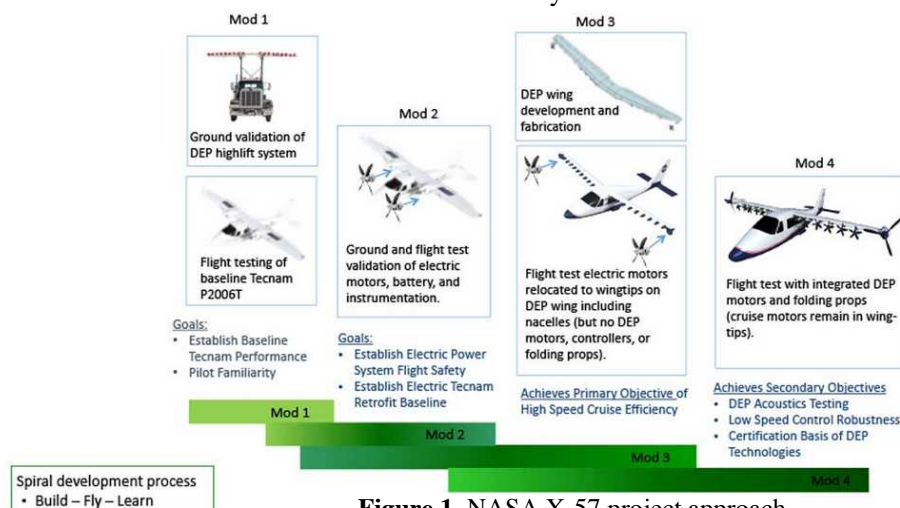


Figure 1. NASA X-57 project approach

(NASA: Borer et al. 2016)

This paper describes the development and implementation of a flexible, modular model architecture for hybrid-electric aircraft.

1.1 Model Architecture Application

Electrically propelled and powered aircraft represent a disruptive technology that, from an aviation safety and reliability perspective, requires an evolutionary technological development path. To support this development, the NRC has initiated two parallel ‘sister’ projects that will provide both a virtual and physical test platform for hybrid-electric aircraft system innovation. The first is a Sustainable Mobility Systems (SiMS) Virtual Demonstrator – a multi-disciplinary virtual prototyping tool that will support the development of hybrid-electric aircraft throughout the system development life cycle. This tool will be used to support the evaluation of individual and integrated component-level technologies associated with hybrid-electric aircraft, including power generation, storage, transmission, conversion, and consumption components. It will also provide support for system and component characterization, optimization, trade studies, performance evaluation, and failure modes and effects analysis. The second is the development of a Hybrid Electric Aircraft Testbed (HEAT) – an airborne electric propulsion test-bed demonstrator that will be used to evaluate various hybrid-electric propulsion systems / configurations and gather experimental data to inform evolving certification requirements. Initially, SiMS will act as a virtual prototyping tool for the design and development of the HEAT physical demonstrator. In turn, HEAT will provide data to validate or tune system and component level models used by SiMS. Together, these systems will provide a dedicated research platform to advance hybrid-electric aircraft development and certification.

1.2 Developmental Approach

For the creation of SiMS, a custom aircraft model architecture was developed in Modelica using a suite of Modelon libraries: Aircraft Dynamics Library, Electrification Library, and Liquid Cooling Library (Modelon AB, 2018). To define a reasonable starting point for the modeling framework, the architecture was established using a real-world electric aircraft: the NASA X-57. The high availability of technical data related to the NASA X-57 project provided a unique opportunity to demonstrate the effectiveness of these libraries in supporting the development of a custom electric aircraft model architecture. In addition, since the X-57 is being developed in stages (Tecnam, 2018), it provided the opportunity to demonstrate the ability of the model architecture to adapt to configuration changes starting from a conventional baseline aircraft – the Tecnam P2006T (Mod 1) and leading to a fully

modified electric aircraft (Mods 2 & 3). Note that the X-57 is currently in development with flight testing of Mod 3 targeted for mid-2020. A model of the final configuration (Mod 4) was not developed.

2 Model Libraries

The Aircraft Dynamics Library from Modelon is a new library offering that includes a full, flexible model architecture and compatible component models for modeling aircraft systems. The library includes a powerful sizing model for implementation of the aircraft systems and can support a range of analyses integrating:

- Specification of aircraft geometry
- Flight dynamics
- Propulsion dynamics
- Aerodynamics
- Energy and fuel consumption
- Fuel dynamics
- Thermal dynamics
- Environmental control
- Auxiliary power
- Actuation systems
- Landing gear
- Flight controls
- Flight maneuvers and mission profiles

The library provides a 6DOF representation of the aircraft flight dynamics using aircraft geometry and the distributed mass and inertia of the individual subsystems (Stengel, 2004). The top level aircraft subsystem decomposition is shown in Figure 2, including the airframe, power systems, and general aircraft systems (including avionics, flight deck and cabin systems, etc). These subsystem models can be further broken down into individual components.

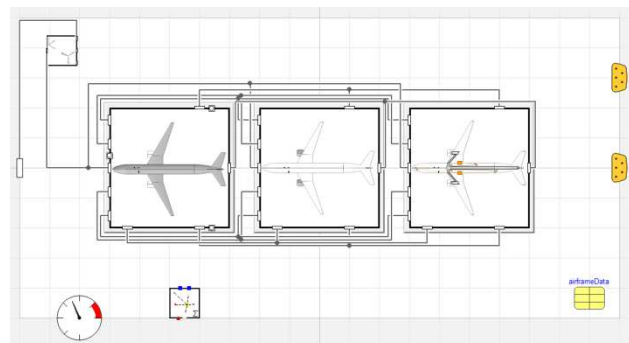


Figure 2. Aircraft decomposition in Aircraft Dynamics Library

Figure 3 shows the next level of decomposition of the airframe subsystem where the fuselage, wings, tail, landing gear, and aerodynamics are specified.

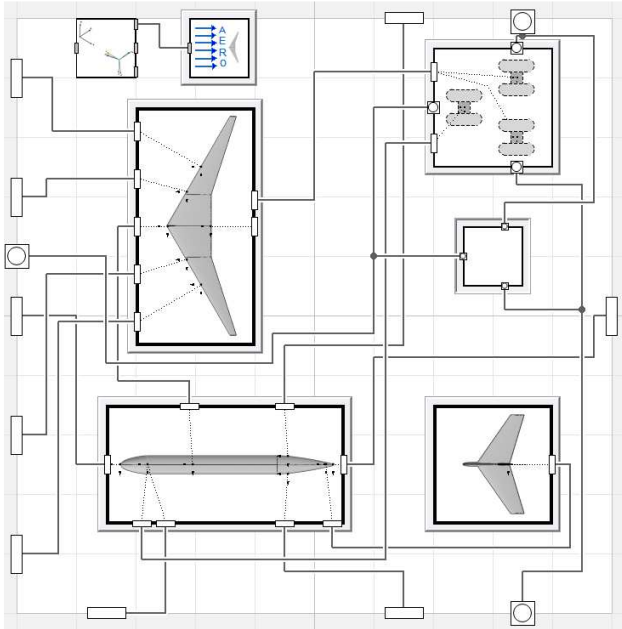


Figure 3. Architecture of the airframe subsystem

Figure 4 shows the architecture of the power subsystem, where the Electrification and Liquid Cooling libraries can be applied, and where the engines, electrical systems, actuation systems, auxiliary power systems and fuel systems are specified.

The sizing model (Kroo, 2001) enables the realization of different types of aircraft quickly. Users are provided with configurable options for input of the different parameters needed to specify the aircraft geometry. This capability allows the model to estimate missing information where minimal data is available (in order to quickly establish a working aircraft model), while also providing options for input of detailed geometry, where this data is known.

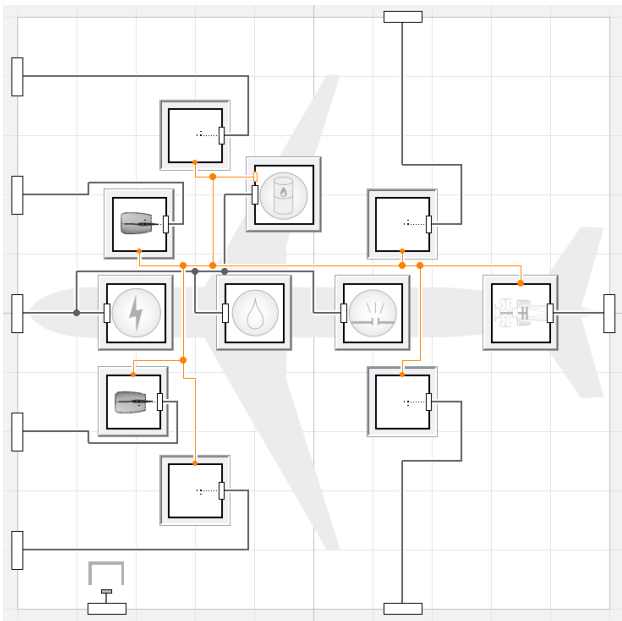


Figure 4. Architecture of the power subsystem

Models are built by configuring templates, and models of different levels of fidelity can easily be implemented within the flexible model architectures. Further details of model implementation will be provided in the following sections describing the implementation of the Tecnam P2006T and X-57 models. Note that the Aircraft Dynamics Library integrates with other Modelon libraries including Jet Propulsion Library, Fuel Systems Library, Environmental Control Library, Hydraulics Library, Pneumatics Library, and Liquid Cooling Library.

3 X-57 Modeling and Simulation

This section describes the development and implementation of the P2006T and X-57 models. The development of these models roughly followed the project approach shown in Figure 1 with the initial development of the baseline P2006T model (Mod 1), followed by various developmental milestones leading to the X-57 (up to Mods 2 & 3).

3.1 Aircraft Overview



(a) Tecnam P2006T



(b) NASA X-57 rendering

Figure 5. Tecnam P2006T and NASA X-57 aircraft

The Tecnam P2006T is a twin-engine four-seat general aviation aircraft. Shown in Figure 5a, the P2006T has a

high wing and twin Rotax 912 S3 100 hp four cylinder internal combustion engines. The engines drive two-bladed constant speed propellers capable of full feathering. Nominal specifications include a maximum cruise speed of 77 m/s (150 kt), 1239 km range, and fuel consumption of 17 L/h (4.5 gal/hr) per engine (Tecnam, 2018).

The NASA X-57 aircraft is a heavily modified Tecnam P2006T and is being developed as part of the Leading Edge Asynchronous Propeller Technology (LEAPTech) project (NASA, 2018), initiated in 2014. Shown in Figure 5b as an artist’s rendering, the X-57 features two wing tip cruise motors and six small electric motors distributed along the leading edge of each wing. The wing of the X-57 is optimized for the high-speed cruise condition, and therefore, its area is substantially reduced and aspect ratio increased compared to the original Tecnam P2006T. In order to produce the necessary lift at the lower speeds of take-off and landing, a trailing edge flap is deflected, and the motors distributed along the leading edge are operated to increase the flow velocity over the wing and therefore increase lift. These lift augmentation propellers fold snugly along the motor nacelles when not in use during cruise to reduce drag.

3.2 P2006T Modeling and Results

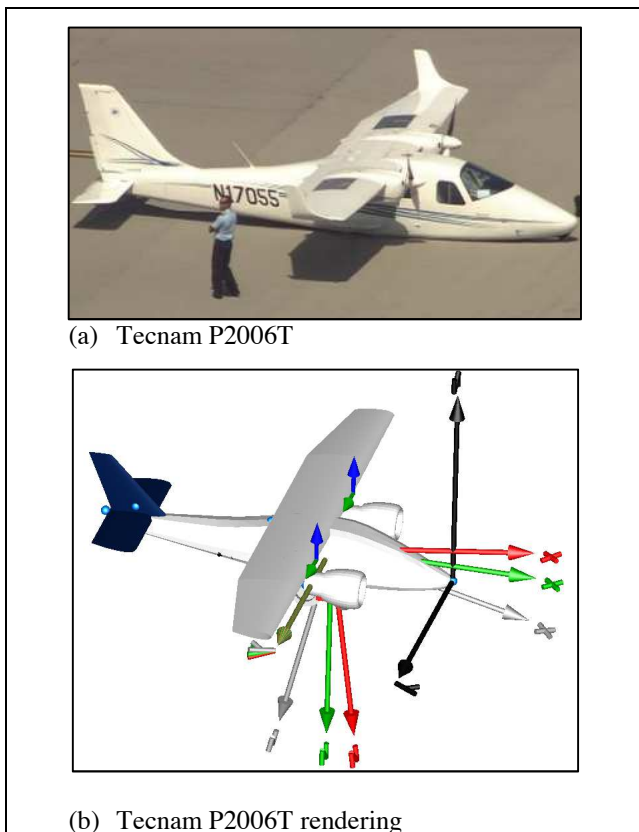


Figure 6. Actual and animation of P2006T aircraft
 The first step in the development of the X-57 Modelica model was the creation of a baseline model of the P2006T aircraft (Mod 1). The sizing model from the

Aircraft Dynamics Library was used to provide reasonable implementations of the various aircraft subsystems for modeling the airframe. A simplified, high level model of the P2006T powertrain was assembled, and a surrogate turboprop model was used to drive the P2006T. A default aerodynamic model provided by the sizing model was used to provide first cut estimates for the aerodynamic forces based purely on the geometry (using information and technical drawings from the manufacturer’s website (Tecnam, 2018)).

The ability to animate the entire aircraft is natively integrated into the individual airframe models generated by the sizing model (see Figure 6). This animation can provide feedback on the aircraft dynamics as well as visual feedback on the geometric specification of the aircraft. Users can then leverage the animation to refine the geometric parameters provided directly or entered into the sizing model.

Figure 7 shows the top level P2006T modeling experiment. This experiment contains the aircraft model and a controller implementation that controls the aircraft elevator and thrust to provide the specified aircraft velocity and altitude profiles. The simulation was run with a cruise velocity set-point of 70 m/s (~136 knots). The resulting aircraft response with respect to velocity, height, drag and lift coefficients, and thrust force are shown in Figure 8.

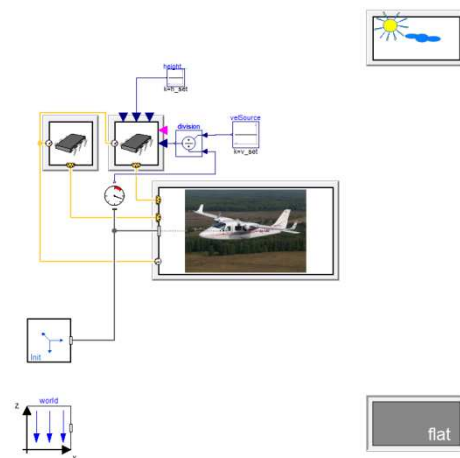


Figure 7. P2006T flight experiment

The simulation resulted in a shaft power requirement (thrust force x velocity) of 124 kW (167 hp), which the combined shaft power of the Tecnam’s two Rotax power plants is capable of producing (150 kW or 200 hp); the aircraft drag is implicit in the power requirement and is therefore also deemed acceptable. The lift coefficient of $C_L = \sim 0.5$ is a reasonable value for an aircraft of this type and is consistent with the lift coefficient obtained in cruise condition under test (Nicolosi, 2010).

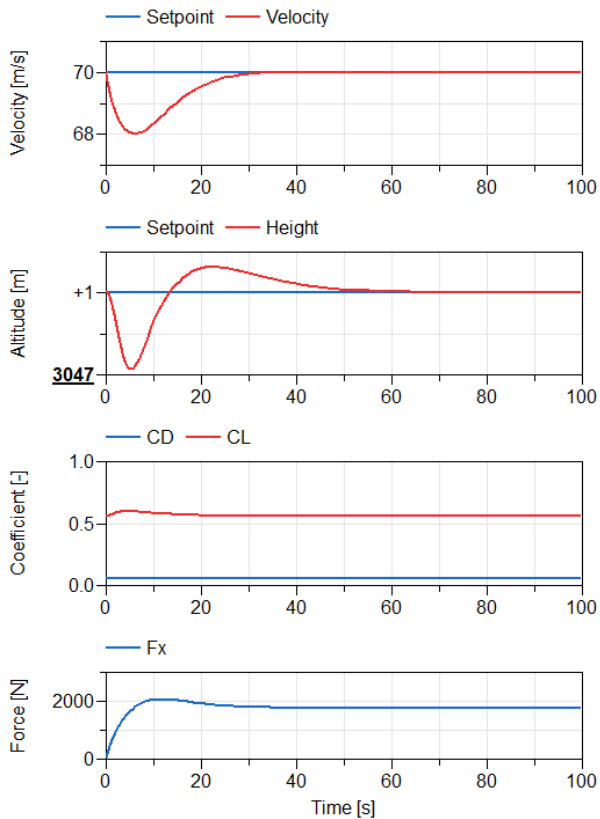


Figure 8. P2006T simulation results

3.3 X-57 Modeling and Results

With a reasonable baseline model for the P2006T achieved, the model was adapted to represent the X-57 in a series of model-based development steps. The modeling steps that parallel the project approach shown in Figure 1 are described in this section.

X-57 Mod 2 involves the development and testing of the electric powertrain, first on the ground and then integrated into the P2006T airframe. The X-57 electric powertrain was modeled using the Electrification Library and cooling system components from Liquid Cooling Library (Modelon AB, 2018). The modeling approach is similar to that described in (Falck, 2017), including thermal models to capture thermal interactions and potential thermal operating constraints for the electric powertrain. These models are focused on overall energy efficiency and thus use an averaged approach for the power electronics (detailed models of electrical system operation with switching is not required). This simplification also improves the computational efficiency of simulations performed over an entire mission profile. Though the specification of the X-57 powertrain is still under development, the electric powertrain model includes the following:

- Dual 120 Ah battery packs with 128 cells in series, 40 cells in parallel
- Two 60 kW electric motors
- Power inverters
- Lossy electric cables

- Two 1.5m propellers
- Air cooled thermal system with flow from motor to inverter

Figure 9 shows one half of the X-57 electric powertrain on a test bench, including the thermal system and propeller. The aircraft speed and ambient air density are applied as boundary conditions, and the motor torque command and propeller pitch angle are applied as commands. The propeller performance is modeled after (Wainausky, 1989).

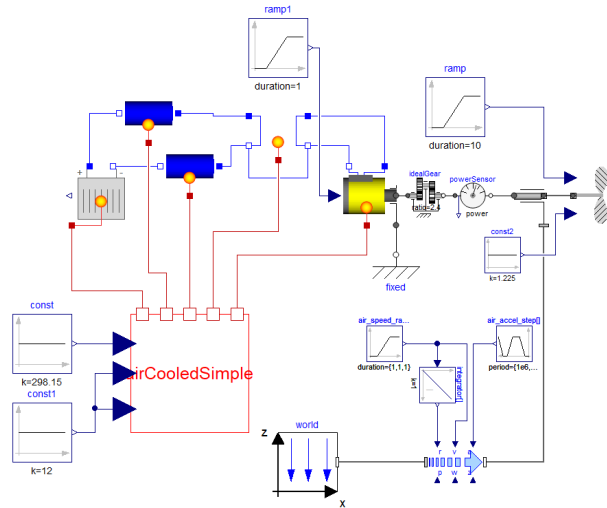


Figure 9. X-57 electric powertrain test bench

This test bench allows for model-based development of the electric powertrain for system sizing and integration, energy consumption, and controls development.

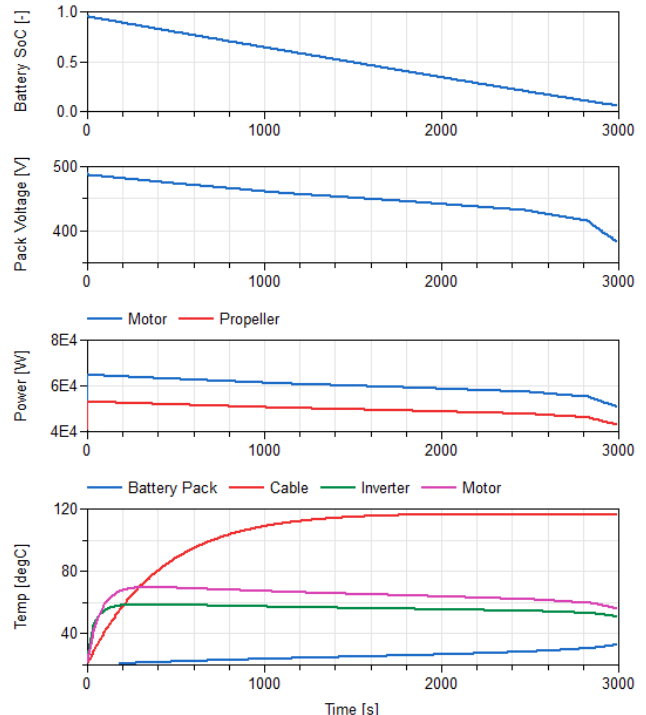


Figure 10. Full battery discharge, full power, V=50 m/s

Figure 10 shows results from the powertrain test bench. These simulations were run with the aircraft speed set at 50 m/s and full power commanded from the motor. This simulation showed that after 3000 s, the battery state of charge went from 0.95 to 0.06. The pack voltage drops as the state of charge decreases due to changes in the battery voltage capability and the increase in resistance; thus, the power delivered by the motor and power produced by the propeller also drop. The battery pack temperature increases over the entire simulation due to the large thermal capacitance of the battery pack. When the state of charge drops, there is more heat generated by the battery pack due to increased electrical resistance and thus an increase in the temperature rise rate. Temperatures for the cable, inverter, and motor are also shown. This type of simulation can be used for system sizing and for estimating potential aircraft range for a specific electric powertrain configuration.

Figure 11 shows results from a blade pitch angle ramp at full power command, with aircraft speed at 50 m/s. As the blade pitch angle increases, the propeller power drops to maintain the constrained air speed condition. The plot shows the power out of the battery pack, delivered by the motor at the shaft, and delivered by the propeller as thrust power.

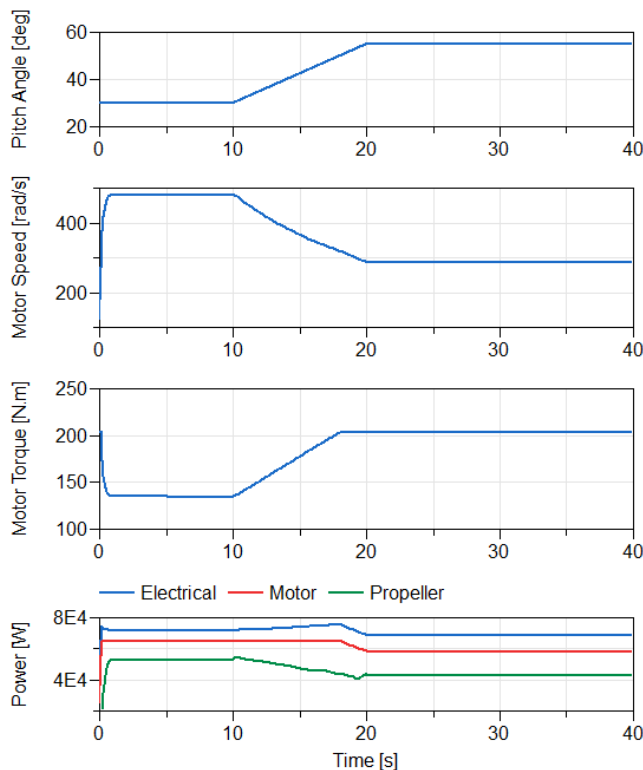


Figure 11. Propeller blade angle ramp, full power command, aircraft speed 50 m/s

This type of simulation can be used to develop optimum system operation strategies for different aircraft operating conditions and power demands.

To facilitate integration of the electric powertrain into the aircraft model, a new power template for Aircraft Dynamics Library was created, as shown in Figure 12. As compared to the original power template shown in Figure 4, this new power template incorporates the following changes to allow drop-in replacement of the various electric powertrain subsystems without requiring any additional connections between subsystems:

- Addition of electrical bus connectors to the engine and electric systems (blue connections)
- Addition of thermal bus connectors to the engine and electric systems (red connections)
- Addition of a new thermal subsystem with electrical and thermal bus connectors

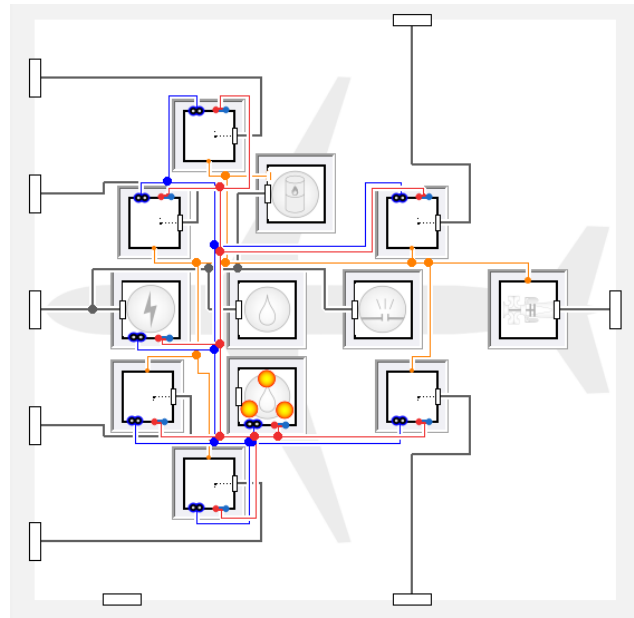


Figure 12. New Power template for electric aircraft

As described in Figure 1, Mod 2 progresses from ground testing of the electric powertrain to integration of the electric powertrain into the P2006T airframe to replace the conventional engine powertrain. Using the new power template shown in Figure 12, the electric powertrain shown in Figure 9 was integrated as follows:

- *electric* subsystem contains the battery pack (1 for each side) for the cruise motors
- *engine2* and *engine3* subsystems contain the cable, inverter, motor, and propeller for the cruise motors (1 for each side)
- *thermal* subsystem provides cooling to components (one thermal system in total to more easily handle the connections between systems on the thermal side)

With the power system implemented, a simple redeclare was all that was needed to create the

model variant of the P2006T with the electric powertrain as shown in Figure 13.

After the initial testing of the electric powertrain in the P2006T airframe, Mod 3 in the X-57 development plan includes the modification of the P2006T airframe to integrate the Distributed Electric Propulsion (DEP) wing design. These same changes were implemented in the model to develop the X-57 Mod 3 aircraft.

```

model P2006TElectric_Aircraft "P2006T with electric powertrain"
  extends X57.Aircraft.TecnamP2006T(redeclare replaceable
    X57_Power power);
end P2006TElectric_Aircraft;

```

Figure 13. Modelica code for electric P2006T aircraft

The final X-57 configuration will include the distributed electric propulsion system and a cruise-optimized wing geometry with a program goal of a 5x reduction in cruise energy consumption when compared with the baseline P2006T (Deere, 2017). For the purposes of the X-57 model in this work, only the cruise motors were considered, as the Mod 3 focus is on understanding energy usage under cruise conditions when the high lift motors are not active. The new wing geometry was implemented based on the geometric specification in (Deere, 2017). The extensive CFD analysis conducted on the proposed X-57 wing designs (Deere, 2017) was used to create a custom aerodynamics model using the Aircraft Dynamics Library. This new table-based lift and drag model was implemented with coefficients as a function of angle of attack. The X-57 aircraft model was then constructed as a variant of the P2006T by redeclare of the airframe, aerodynamics model, and power subsystem as shown in Figure 14.

```

model X57_Aircraft "X57 aircraft"
  extends X57.Aircraft.TecnamP2006T(redeclare replaceable
    X57_Power power,
    redeclare X57.Airframe.AirframeX57 airframe(redeclare
    X57TableBasedAero aero));
end X57_Aircraft;

```

Figure 14. Modelica code for X-57 aircraft

The animation for the X-57 aircraft is shown in Figure 15. Note the visual depiction of the new wing design as compared to the P2006T in Figure 6. The P2006T wing area is reported as 159 ft² with an aspect ratio of 8.8 as compared to the simulated X-57 design with wing area of 67 ft² and an aspect ratio of 15.

The X-57 Mod 3 flight experiment is shown in Figure 16. The model includes the X-57 Mod 3 aircraft and a controller. The controller acts on motor torque, propeller blade angle, and aircraft elevator to achieve a desired height, propeller speed, and aircraft speed. Simple PID controllers were sufficient for targeting cruise operating conditions.

Figure 17 shows results from a velocity ramp for the X-57, from 60 m/s to 75 m/s with the propeller speed command at a constant 2250 RPM. As expected, for a

constant speed propeller, the pitch angle and the battery pack, motor, and propeller power increase with an increase in aircraft velocity.

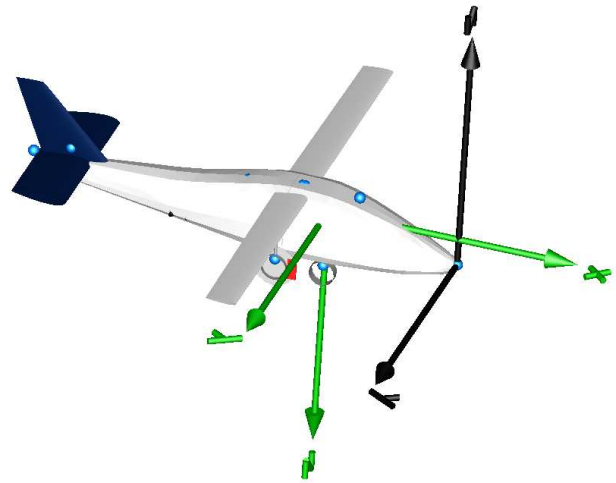


Figure 15. X-57 animation

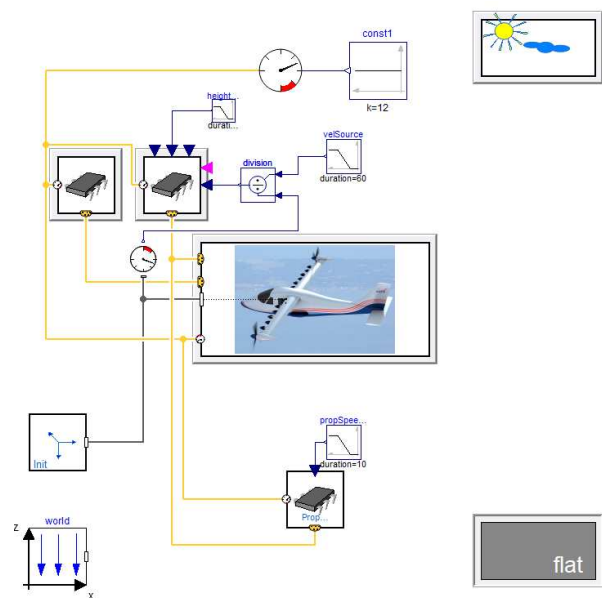


Figure 16. X-57 flight experiment

Figure 18 shows the sensitivity of the battery state of charge to the drag coefficient for the X-57 wing design with the same velocity ramp simulation shown in Figure 17. The variations in drag coefficient were achieved via a multiplier of 1, 1.1, 1.25, 1.4, and 1.5 on the table-based drag values.

Figure 19 shows the sensitivity of the power consumption to the aircraft altitude using the same velocity ramp conditions simulated previously. The standard atmosphere model in Aircraft Dynamics Library provides atmospheric conditions that vary with altitude. These simulations were run at altitudes of 3048 m, 1500 m, and 250 m, where the results are primarily driven by the increase in density at lower

altitudes and thus, higher drag. Because the lift is also affected by the density, there is an interaction with the aircraft angle of attack, an effect that is not seen when varying the drag coefficient in Figure 18.

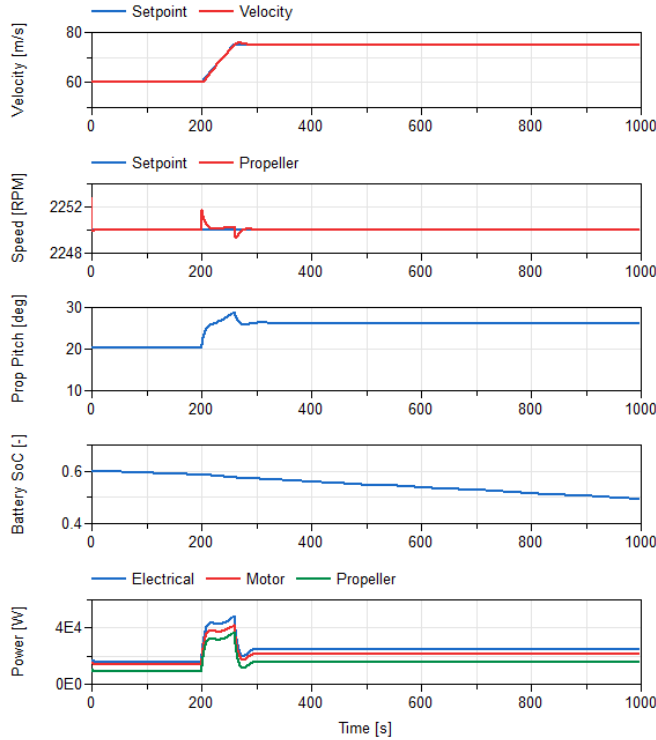


Figure 17. X-77 velocity ramp, constant height

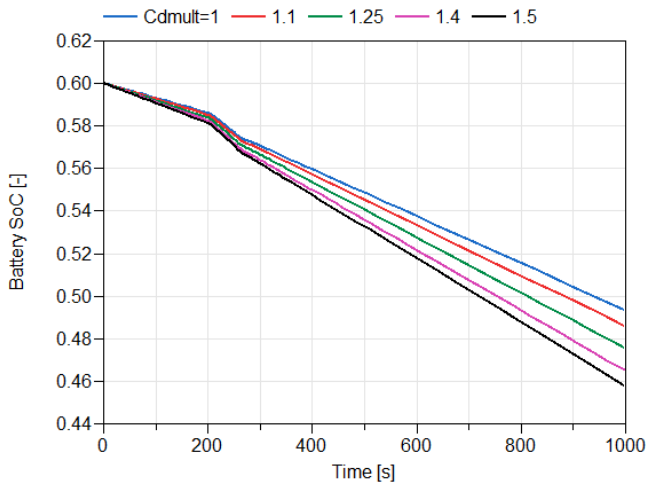


Figure 18. Battery state of charge sensitivity to drag with velocity ramp

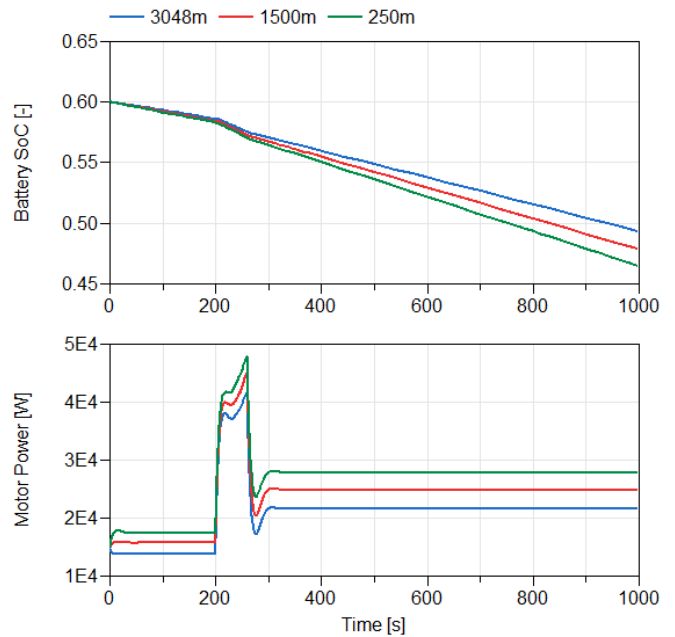


Figure 19. Power consumption sensitivity to aircraft altitude with velocity ramp

4 Summary

The move towards ‘more electric’ and ‘fully electric aircraft’ has significantly expanded the design space for aircraft systems. In order to support this paradigm shift in aircraft design, a multi-physics modeling approach that captures the relevant mechanical, electrical, and thermal dynamics was undertaken. This approach provides the framework for rapid model variant development and for performing a range of analyses of multi-domain systems. Analyses ranging from electric powertrain test benches to integrated electric aircraft performance were performed using a coordinated suite of Modelon libraries, focusing on Mod 1 to Mod 3 of the X-77 project for cruise efficiency demonstration. The Mod 3 model variant can be extended in future work to include the fully distributed high lift electric propulsion system. With a full model, an extended range of mission profiles, including takeoff and landing can be simulated.

As a result of this work, the underlying framework, power train model architecture, and new electrification model templates have been developed and demonstrated, and will be used to support the continued advancement of the virtual (SiMS) and physical (HEAT) demonstrator systems.

Acknowledgements

The authors would like to thank Jim Claesson for his support with the library releases for this effort and Abhilash Kumar for his work on these models.

References

- Boeing, (2013). <http://787updates.newairplane.com/787-Electrical-Systems/787-electrical-system>
- Borer, N. K., Patterson, M. D., Viken, J. K., Moore, M. D., Clarke, S., Redifer, M., Christie, R., Stoll, A., Dubois, A., Bevirt, J., Gibson, A., Foster, T., Osterkamp, P., “Design and Performance of the NASA SCEPTOR Distributed Electric Propulsion Flight Demonstrator,” *AIAA-2016-3920, 16th AIAA Aviation Technology, Integration, and Operations Conference, AIAA AVIATION Forum*, Washington, D.C., June 2016.
- Deere, K.A., Viken, J.K., Viken, S.A., Carter, M.B., Wiese, M.R. and Farr, N., “Computational Analysis of a Wing Designed for the X-57 Distributed Electric Propulsion Aircraft”, *AIAA Aviation Forum*, Denver, CO, June 5-9, 2017.
- E-Fan / E-Fan X (2017). “Airbus, Rolls-Royce and Siemens develops Hybrid-Electric demonstrator”, <https://leehamnews.com/2017/11/29/airbus-rolls-royce-siemens-develops-hybrid-electric-demonstrator/>
- Falck, R. D., Chin, J. C., Schnulo, S. L., Burt, J. M., and Gray, J. S., “Trajectory optimization of electric aircraft subject to subsystem thermal constraints,” *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Denver, CO, 2017.
- Guy Norris. ["Motor Mounting Marks Milestone for NASA's Electric X-plane"](#). *Aviation Week & Space Technology*, Sep 5, 2018.
- Kroo, I., Shevell, R., “Aircraft Design: Synthesis and Analysis”, *Desktop Aeronautics*, 2001.
- Modelon AB, Lund, Sweden. (2018). *Aircraft Dynamics Library*. <http://www.modelon.com/products/modelon-library-suite/aircraft-dynamics-library/>
- Modelon AB, Lund, Sweden. (2018). *Electrification Library*. <http://www.modelon.com/products/modelon-library-suite/electrification-library/>
- Modelon AB, Lund, Sweden. (2018). *Liquid Cooling Library*. <http://www.modelon.com/products/modelon-library-suite/liquid-cooling-library/>
- NASA, 2018, “NASA Armstrong Fact Sheet: NASA X-57 Maxwell”. <https://www.nasa.gov/centers/armstrong/news/FactSheets/FS-109.html>
- Nicolosi, F., De Marco, A., Vecchia, P.D., “Stability, Flying Qualities and Parameter Estimation of a Twin-Engine CS-23/FAR 23 Certified Light Aircraft”, *AIAA Guidance, Navigation, and Control Conference*, Toronto, ON, Aug 2-5, 2010.
- Pipistrel, (2018). <https://www.pipistrel.si/plane/alpha-electro/overview>
- Stengel, R. F., “*Flight Dynamics*”, Princeton University Press, 2004.
- Taylor, John W. R., “Jane's All the World's Aircraft” *Page 573, London: Jane's Yearbooks*, 1974-75.
- Tecnam, (2018). “P2006T Homepage”, <http://www.tecnam.com/aircraft/p2006t/>
- Tecnam, (2018). “ P2006T X-57 MAXWELL NASA Homepage”, <https://www.tecnam.com/innovation/p2006t-x-57-maxwell-nasa/>
- Wainausky, H. S., Rohrbach, C., Wynosky, T. A., “Prop-Fan Performance Terminology”, *SAE Aerospace Technology Conference and Exposition*, Long Beach, CA, Oct 5-8, 1987.

A Modelica Library for Spacecraft Thermal Analysis

Tobias Posielek¹

¹Institute of System Dynamics and Control, DLR German Aerospace Center, Oberpfaffenhofen, Germany
tobias.posielek@dlr.de

Abstract

In spacecraft missions it is vital to maintain all spacecraft components within their required temperature limits. Thus, a model incorporating all main heat fluxes acting on the spacecraft is necessary to allow for the design of a thermal control subsystem. This paper introduces the thermal space systems library which implements common models of radiation and thermal components of a spacecraft. Special effort is put into the calculations of the angles describing the orientation of the spacecraft with respect to sun and earth. Issues occurring due to the recalculation of the angles in each time step are shown and methods for their determinations are given.

Keywords: space modeling, thermal modeling, angle determination

1 Introduction

In spacecraft engineering, it is essential to ensure that all components operate in their appropriate temperature range to avoid malfunction and equipment breakage. Therefore, an analysis of the thermal dynamics is a necessity to design the required thermal control (Gilmore and Bello 1994), (Meseguer, Pérez-Grande, and Sanz-Andrés 2012), (Fortescue, Swinerd, and Stark 2011). A rigorous description of the thermal system is difficult as it has to incorporate the orbit and the orientation of the spacecraft during the mission, as well as the sun's position and the dissipated energy within the spacecraft. The modelling of the thermal system is a present topic of interest (Ruan, Hu, and Sun 2017) (Lefeng et al. 2017) (Qian et al. 2015). Approaches of various complexity exist to design the thermal control. Simple design approaches consider only static worst case scenarios to account for degradation and orbit thermal dynamics (Larson and Wertz 1991). Other methods use analytical models to obtain the dynamic evolution of the temperature over the course of multiple orbits (Tsai 2004).

The proposed library allows the simulation of the complete spacecraft system including the thermal system as well as the electric and mechanical system providing the dissipated energy and spacecraft orientation dependent on the spacecraft mission. The library is proposed in view of simple analytical mod-

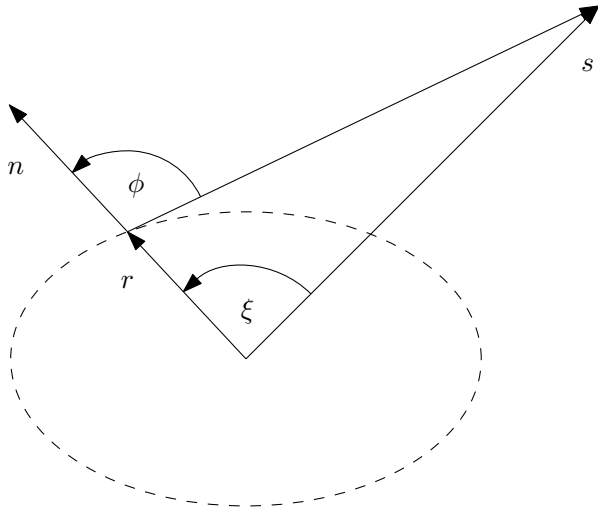
els. Generally, a spacecraft is modelled by a huge number of nodes with different heat fluxes acting on each. We will only model the most important nodes e.g. each surface may be modelled as a node for a cuboid spacecraft. For each of these nodes the temperature dynamic is determined by the dynamic of its adjacent nodes and the four main heat flows due to the environment. One main point which will be illuminated is the calculation of the angle between the spacecraft surfaces and its surroundings. As the attitude of spacecraft is usually not known a priori and determined online, suitable methods to calculate this angle are proposed. The library is created in view of earth orbiting spacecraft. However, the library can also be used for simulations of spacecraft leaving earth orbit as long as modifications regarding the coordinate systems and approximations, such as shadow calculations, are made. The proposed library uses the other Modelica-based libraries of the Institute of System Dynamics and Control at the DLR German Aerospace Center such as the Environment library (Briese, Klöckner, and M. Reiner 2017) and SpaceSystems library (M. J. Reiner and Bals 2014). The library is created as an in-house library as a part of the design of an energy management for spacecraft. Section 2 introduces the essential fundamentals for the thermal dynamics. Section 3 gives details to the Modelica implementation and in Section 4 an example scenario is simulated to show the functionality of this library.

2 Fundamentals

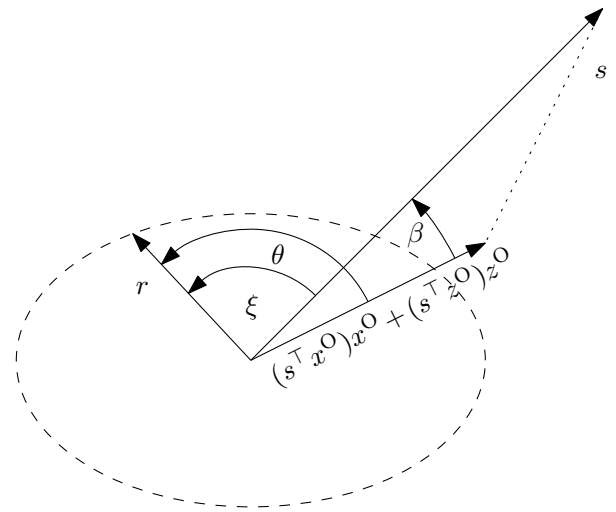
This section introduces the coordinate systems, heat fluxes, solar angles, form factor and shadow function necessary to simulate the spacecraft thermal system.

2.1 Coordinate Systems

The Earth-Centered Inertial (ECI) Frame is defined such that the x^1 -axis points in direction of vernal equinox, this is the intersection between the equator and the sun's apparent orbit during spring. The z^1 -axis is parallel to the mean Earth's rotation axis and towards the North Pole and the y^1 -axis completes right handed coordinate system. For all following reference frames the rotation matrix to ECI coordinates is given by their coordinate axes. Each coordinate system will be denoted with a superscript which will



(a) Main angles influencing the temperature evolution in a spacecraft. The solar zenith angle ξ is defined as the angle between the vector to the spacecraft r and the vector pointing to the sun s . The normal solar angle ϕ describes the angle between the normal of a spacecraft surface n and the vector pointing from spacecraft to the sun. The dotted line describes the spacecraft orbit.



(b) Angles describing the influence of the solar zenith angle ξ . The solar noon angle θ describes the angle between the vector pointing to the spacecraft r and the solar noon, i.e. the vector pointing to the sun projected on the orbit plane $(x^O s)x^O + (z^O s)z^O$. The beta angle β is defined as the angle between the orbit plane and the vector pointing to the sun s .

Figure 1. Solar Angles

be used for the notation of their coordinate axes and rotation matrices. We denote $T^{S,I} = [x^S \ y^S \ z^S]^T$ as the transformation from ECI coordinates to an arbitrary coordinate system with superscript S . So for r^I in ECI coordinates the transformed vector r^S is calculated via

$$r^S = T^{S,I} r^I. \quad (1)$$

The orbit frame is defined for a spacecraft in an elliptical orbit with position $r^I(t)$ and velocity $v^I(t)$ in inertial coordinates by the y^O -axis which is normal to the orbit plane in direction of negative angular momentum, the z^O -axis which points to geocentric nadir and the x^O -axis which completes the right handed coordinate system and is for circular orbits in direction of velocity. We omit the time argument on the right hand side and obtain the transformation from ECI coordinates to orbit coordinates as

$$T^{O,I}(t) = \begin{bmatrix} \frac{r^I \times (r^I \times v^I)}{\|r^I \times (r^I \times v^I)\|} & -\frac{r^I \times v^I}{\|r^I \times v^I\|} & -\frac{r^I(t)}{\|r^I(t)\|} \end{bmatrix}^T. \quad (2)$$

2.2 Environmental Heat Fluxes

Mainly four environmental heat fluxes are acting on a spacecraft surface, namely the heat flux due to direct solar irradiation, the solar radiation reflected by the earth, the radiation of the earth emitted in the infra-red spectrum and the radiation of the spacecraft emitted to deep space (Larson and Wertz 1991),(Meseguer, Pérez-Grande, and Sanz-Andrés 2012). Each of these fluxes and its calculation is introduced in this section.

2.2.1 Direct Solar

The solar radiation is the main factor influencing temperature changes of the spacecraft. A solar constant G_{s0} is defined as in (Meseguer, Pérez-Grande, and Sanz-Andrés 2012) which gives the mean solar irradiance acting on a unit area perpendicular to the solar rays in a distance of 1 ua where ua denotes the astronomical unit. As the amount of irradiance crossing spherical surfaces with different radii is assumed to be constant, the solar irradiation G_s scales with distance as

$$G_s(d) = G_{s0} \frac{d_0^2}{d}$$

where d is the distance in astronomical units and $d_0 = 1 \text{ ua}$. The solar energy is mostly distributed in visual and short wavelength infra-red (Larson and Wertz 1991). This allows for surfaces which are very reflective in the solar spectrum but highly emissive to long wavelength infra-red. A simple analytical model incorporates the angle $\phi = \phi(n, r, s) \in [0, \pi]$ between the surface normal and the sun and the shadow of the earth described by the shadow coefficient $\nu = \nu(r, s) \in [0, 1]$ introduced in Section 2.5. Then the acting solar flux reads

$$Q_{\text{sun}} = \begin{cases} \alpha G_s \left(\frac{\|s-r\|}{1 \text{ ua}} \right) A \cos(\phi) \nu & \text{if } 0 < \phi < \frac{\pi}{2} \\ 0 & \text{if } \frac{\pi}{2} < \phi < \pi \end{cases} \quad (3)$$

where α denotes the solar absorptance of the surface and A the area of the surface.

2.2.2 Albedo

By albedo we denote the part of the solar radiation which is reflected by the earth or scattered by the planet surface and atmosphere. Combining the simple models from (Larson and Wertz 1991) and (Meseguer, Pérez-Grande, and Sanz-Andrés 2012) we obtain

$$Q^{\text{alb}} = \begin{cases} \rho_{\text{alb}} \alpha G_s(d) A F_{\text{form}} \cos(\xi) & \text{if } 0 < \xi < \frac{\pi}{2} \\ 0 & \text{if } \frac{\pi}{2} < \xi < \pi \end{cases} \quad (4)$$

where $\rho_{\text{albedo}} \in [0, 1]$ is the albedo coefficient. This coefficient can vary over the course of an orbit and depends on the orbits inclination. This model incorporates the solar zenith angle $\xi(s, r) \in [0, \pi]$, the angle between the sun and the spacecraft, and a form factor $F_{\text{form}}(r, n)$ defined in Section 2.4 to describe the part of the radiation that actually strikes the spacecraft surface.

2.2.3 Planetary Radiation

As planetary radiation we denote the thermal radiation which is emitted by the planet as long wavelength infra-red radiation. The emitted radiation can be calculated as the absorbed solar radiation of the planet minus the radiation emitted via albedo. Then, by assuming the planet to be a black body we obtain the planetary infra-red thermal heat acting on a surface of a spacecraft as in (Meseguer, Pérez-Grande, and Sanz-Andrés 2012)

$$Q^{\text{planet}}(r, n) = \varepsilon A F_{\text{form}}(r, n) \sigma T_p^4 \quad (5)$$

where T_p denotes the black body temperature of the earth, σ the Stefan-Boltzmann constant and ε the infra-red emissivity of the surface. Instead of using the black body temperature of the earth (5), it is often written as

$$Q^{\text{planet}}(r, n) = \varepsilon A F_{\text{form}}(r, n) I_{\text{IR}} \quad (6)$$

where I_{IR} is the intensity of earth infra-red flux to account for the variation of Q^{planet} . Note that I_{IR} is actually not a constant but also varying over the course of an orbit. However, the variation of I_{IR} is small in comparison to the albedo variation.

2.2.4 Radiation to Deep Space

The outer surfaces of a spacecraft are radiatively coupled to space. The energy of the reradiation to space is usually in the long wave infra-red spectrum and can be described by

$$Q^{\text{ds}} = \varepsilon A \sigma T^4 \quad (7)$$

where T denotes the temperature of the surface.

These four heat fluxes are the main environmental heat fluxes acting on the spacecraft. Other fluxes due to the environment exist but are neglected in the

analysis due to their minor influence on most spacecraft. Numerical values for the parameters describing the solar absorptivity and infra-red emissivity of different surface can be found in the literature such as (Larson and Wertz 1991). Hot and cold case scenario parameters for ρ_{alb} , I_{IR} and G_s dependent on the orbit and can be found in (Larson and Wertz 1991). Formula to describe the solar angles ϕ , ζ , the form factor F_{form} and the shadow coefficient ν are introduced in the following sections.

2.3 Thermal Angles

For the calculation of solar, albedo and infra-red irradiation, different angles describing the position of the sun and the attitude of the surface are of interest. These angles are visualized in Figure 1. Figure 1a shows the zenith angle ξ and normal solar angle ϕ which are the angles influencing the generation of heat and Figure 1b the solar noon angle θ and beta angle β which can describe the influence of the solar zenith angle as will be explained later. In this section, we define these angles, show the relations between them and introduce two different ways to calculate these angles.

Problem Formulation

Usually two vectors v_1 and v_2 are given in a reference coordinate system. In order to calculate the angle between these two vectors, it may seem advantageous to use the scalar product as in formula (8) as you do not need any other rotations or coordinates system and use only the property of the scalar product. This however, gives only angles between $[0, \pi]$ which is sufficient for many calculations which use uneven functions but it leads to undesired results when rotations are considered as illustrated in Figure 3. In this figure the angle between the vectors v_1 and $v_2(t)$ is displayed on the left hand side over the course of a full uniform planar rotation illustrated in the middle. The angle moves between 0 and π which makes no unique identification of the position of v_2 from the angle possible. Note that in the control context, the uniqueness issue can be solved by using quaternions which use the normal axes of the rotation as additional information. On the right hand side, the desired angle evolution is displayed which ensures the bijection between position and angle in a single rotation. In order to achieve this angle definition between $(-\pi, \pi]$, we use the properties of cylinder coordinates. Such a definition is simple if a coordinate system is constructed which $x - y$ -plane describes the rotation plane or if the reference frame can be rotated on the rotation plane. Note however, that only a minimum of information about the rotation is known and one can only rely on the current value $v_1(t)$ and $v_2(t)$ but not on a closed description of the functions $v_1(\cdot)$ and $v_2(\cdot)$. This information has to be used to construct

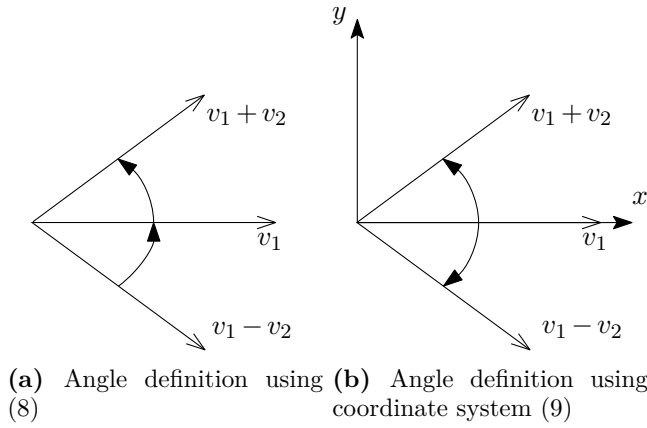


Figure 2. Different angle definitions

the same coordinate system at every time step over the course of the rotation. Problems using intuitive coordinate systems definitions are illustrated in Figure 5 and 4. Thus, an additional vector is necessary to construct this coordinate system. In the case of orbit rotations this vector comes by the cross product of velocity and position.

Angle Definitions

The most intuitive definition defines the angle $\theta \in [0, \pi]$ as the smaller positive angle between the two vectors $v_1, v_2 \in \mathbb{R}^3$ as

$$\theta = \angle(v_1, v_2) := \cos^{-1} \left(\frac{v_1^\top v_2}{\|v_1\| \|v_2\|} \right) \quad (8)$$

where \cos^{-1} denotes the inverse of the cosine with a domain of $[0, \pi]$. This definition is sufficient for most purposes especially if only the cosine of an angle is of interest. However, as a result the angle between v_1 and $\alpha v_1 + v_2$ is the same as between v_1 and $\alpha v_1 - v_2$ with $v_1^\top v_2 = 0$ and $\alpha \in \mathbb{R}$ which is undesirable in view of planar rotations as illustrated in Figure 2a. This flaw can be overcome by using a cartesian coordinate system and its polar coordinate representation.

For cartesian coordinate system with axes x, y and z represented by its transformation matrix $T = [x \ y \ z]$ so that Tv_1 and Tv_2 are in the x - y -plane, we define the angle $\theta \in (-\pi, \pi]$ by

$$\theta = \angle^x(v_1, v_2) := \text{atan2}(e_2 T v_2, e_1 T v_2) - \text{atan2}(e_2 T v_1, e_1 T v_1). \quad (9)$$

where e_i denotes the i -th unit vector in \mathbb{R}^3 and atan2

the extension of the atan function as

$$\text{atan2}(b, a) := \begin{cases} \text{atan} \left(\frac{b}{a} \right) & \text{if } a > 0 \\ \text{atan} \left(\frac{b}{a} \right) + \pi & \text{if } a < 0 \wedge b \geq 0 \\ \text{atan} \left(\frac{b}{a} \right) - \pi & \text{if } a < 0 \wedge b < 0 \\ \frac{\pi}{2} & \text{if } a = 0 \wedge b > 0 \\ -\frac{\pi}{2} & \text{if } a = 0 \wedge b < 0 \\ \text{undefined} & \text{if } a = 0 \wedge b = 0 \end{cases}$$

We use the superscript x in \angle^x to reference to the corresponding x - y - z -coordinate system which describes T . This definition gives for planar rotations angles the results as desired and is illustrated in Figure 2b. The angle between v_1 and $v_1 + v_2$ and v_1 and $v_1 - v_2$ have different signs in comparison to Figure 2a.

With this definition we can describe the angles for planar rotations by using an at the beginning established coordinate system with the mentioned properties. However, as the desired reference coordinate systems for the calculations of the angles are subject to slow changes, it is necessary to redefine the coordinate system at every point of time. This means the coordinate axes have to be constantly recalculated. Clearly, it is desirable to obtain continuous axes that do not experience a change of sign. Furthermore, the coordinate system shall be right handed and use only information about the current point of time.

By the definition of the cross product, it is sufficient to use only two vectors v_1 and v_2 to define a coordinate system via

$$x = \frac{v_1}{\|v_1\|}, \quad (10a)$$

$$y = z \times x, \quad (10b)$$

$$z = \frac{v_1 \times v_2}{\|v_1 \times v_2\|}. \quad (10c)$$

However, for a constant v_1 but a rotating v_2 the y - and z -axis change their direction when v_1 and v_2 become parallel as can be seen in Figure 4.

Consider the Gram Schmidt process as a way to construct the coordinate system with

$$x = \frac{v_1}{\|v_1\|}, \quad (11a)$$

$$y = \frac{v_2 - (x^\top v_2)x}{\|v_2 - (x^\top v_2)x\|}, \quad (11b)$$

$$z = \frac{v_3 - (x^\top v_3)x - (y^\top v_3)y}{\|v_3 - (x^\top v_3)x - (y^\top v_3)y\|}. \quad (11c)$$

However, with this definition it cannot be guaranteed that the resulting coordinate system is right handed as illustrated in Figure 5.

We combine these two methods in order to obtain a continuous right handed coordinate system.

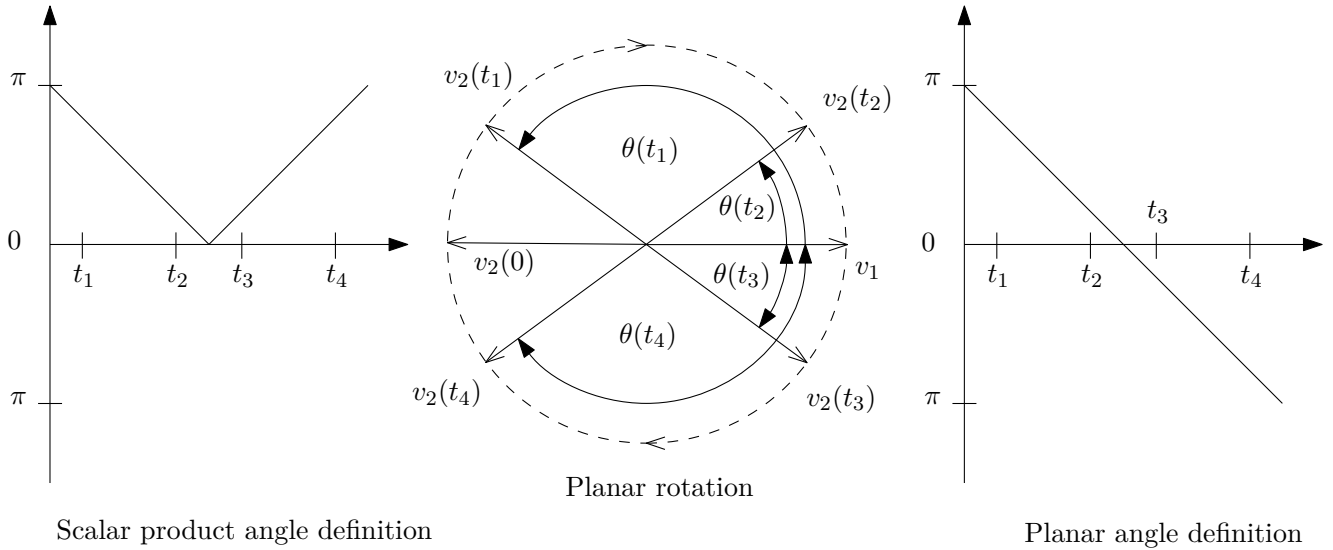


Figure 3. Angle of a planar rotation described by two different definitions

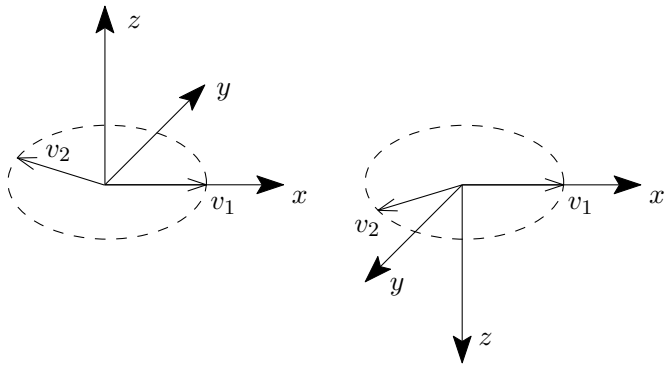


Figure 4. Defined coordinate system using (10)

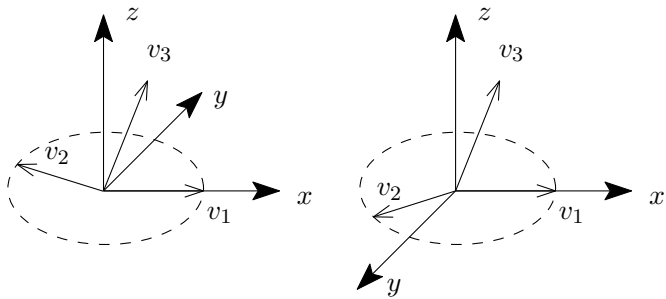


Figure 5. Defined coordinate system using (11)

Let $x^{\text{Gram}}, y^{\text{Gram}}, z^{\text{Gram}}$ be the coordinate axes as in (11). Then define for the coordinate system $T = [x \ y \ z]$ as

$$x = x^{\text{Gram}}, \quad (12a)$$

$$y = z \times x, \quad (12b)$$

$$z = z^{\text{Gram}}. \quad (12c)$$

Thus (9) gives with (11) and (12) a method to calculate the continuous angle between v_1 and v_2 using an additional vector v_3 . This method is introduced in

view of continuous rotations of v_1 and v_2 in a slowly changing v_1 - v_2 -plane. However, it must be ensured that the plane normal does not get perpendicular to v_3 .

2.3.1 The Solar Noon Angle

The solar noon angle θ is the angle between the spacecraft vector r and the sun pointing vector s projected on the orbit plane

$$\theta = \angle^{x^{\text{SN}}} (r, (x^{\text{O}}s)x^{\text{O}} + (z^{\text{O}}s)z^{\text{O}}), \quad (13)$$

using (9) and $\{x^{\text{SN}}, y^{\text{SN}}, z^{\text{SN}}\}$ denoting the coordinate system obtained with Equation (11) and (12) with the vectors $v_1 = (x^{\text{O}}s)x^{\text{O}} + (z^{\text{O}}s)z^{\text{O}}$, $v_2 = r$ and $v_3 = -y^{\text{O}}$. This definition gives for a single orbit of a spacecraft an angle between $(-\pi, \pi]$ with one discontinuity at most.

2.3.2 The Beta Angle

The beta angle $\beta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ defined as in (Meseguer, Pérez-Grande, and Sanz-Andrés 2012) describes the relative orientation of the orbit with regard to the sun, and is defined as the minimum angle between the orbit plane and the solar vector. The beta angle is defined as positive if the spacecraft orbits in a counter clockwise direction and negative if it revolves clockwise with respect to the sun as

$$\beta = \begin{cases} \angle(s, (x^{\text{O}}s)x^{\text{O}} + (z^{\text{O}}s)z^{\text{O}}) & \text{if } s^{\text{T}}y^{\text{O}} < 0 \\ -\angle(s, (x^{\text{O}}s)x^{\text{O}} + (z^{\text{O}}s)z^{\text{O}}) & \text{if } s^{\text{T}}y^{\text{O}} \geq 0 \end{cases} \quad (14)$$

using the definition of the orbit frame from Section 2.1 and Equation (8). Another way to calculate the beta angle is to use the normal of the orbit plane and parameterise the vectors by the orbital elements describing the movement of the sun and the satellite. Consider the sun as a satellite of the earth with the

inclination i^s and the sum of the argument of periapsis and true anomaly $\omega^s + \nu^s$, i.e. the oplicity of the ecliptic and the true solar longitude of the ecliptic. Then the vector to the sun s and the vector orthogonal to the plane y^O can be written in ECI coordinates as:

$$\begin{aligned} s &= \cos(\omega^s + \nu^s)x^I + \sin(\omega^s + \nu^s)\cos(i^s)y^I \\ &\quad + \sin(\omega^s + \nu^s)\sin(i^s)z^I, \\ y^O &= \sin(\Omega)\sin(i)x^I - \cos(\Omega)\sin(i)y^I + \cos(i)z^I. \end{aligned}$$

Instead of calculating the angle to the projection we calculate the angle to the orbit normal as

$$\begin{aligned} \sin(\beta) &= -\cos\left(\beta + \frac{\pi}{2}\right) = -s^\top y^O \\ \Rightarrow \beta &= \sin^{-1}\left(\cos(\omega^s + \nu^s)\sin(\Omega)\sin(i) \right. \\ &\quad \left. - \sin(\omega^s + \nu^s)\cos(i^s)\cos(\Omega)\sin(i) \right. \\ &\quad \left. + \sin(\omega^s + \nu^s)\sin(i^s)\cos(i)\right). \end{aligned} \quad (15)$$

This description emphasises the dependence of the β angle from the orbit inclination and longitude of the ascending node.

2.3.3 The Solar Zenith Angle

The solar zenith angle is defined as in (Meseguer, Pérez-Grande, and Sanz-Andrés 2012) to describe the portion of the illuminated planet which is seen by the spacecraft. The solar zenith angle ξ is defined as the angle between the spacecraft vector r and the sun pointing vector s as

$$\xi = \angle(r, s) \quad (16)$$

using (8). In order to enable a thermal analysis dependent of the orbit attitude, the influence of this angle can be described by the slowly time varying beta angle β and the periodic solar noon angle θ .

For the solar zenith angle ξ , the beta angle β and the solar noon angle θ holds

$$\cos \xi = \cos \beta \cos \theta. \quad (17)$$

As can be seen in Equation (4) the solar zenith angle influences the acting heat significantly. By using (17) we have introduced two different angles which allow analysing the impact of the chosen satellite orbit. The satellite orbit can be described by the six orbital elements a , ε , i , Ω , ω and M_0 . If the orbiting object is only influenced by a gravitation field described by a spherical symmetric planet these orbital elements are constant. In many applications, orbits are chosen to be circular sun synchronous orbits. Thus, a uniform movement is obtained and the solar noon angle can be described as $\theta = \omega_o t$, where ω_o is the angular rotation rate dependent on the semimajor axis a . However, the beta angle is determined by the inclination of the

orbit i and the of the longitude of the ascending node Ω as can be seen in (15). Therefore, the choice of Ω influences the heat acting on the satellite due to the sun significantly.

2.3.4 The Normal Solar Angle

The normal solar angle ϕ is defined between the normal of a spacecraft surface n and the vector pointing to the sun $s - r$ as

$$\phi = \angle(s - r, n) \approx \angle(s, n). \quad (18)$$

This approximation holds because the distance between earth origin and spacecraft is negligible compared to the distance between sun and spacecraft in low earth orbits.

2.4 Form Factor

For the form factor described in the previous section it is sufficient to assume the spacecraft surface to be a infinitesimally small plate and the earth to be a sphere. Then we can use the results from (Juul 1979) and obtain the form factor as a function of distance to the plate and angle $\zeta = \angle(r, n)$, the angle between the normal of the plate n and vector between earth and plate which is approximately the vector between earth and spacecraft r . Let $H = \frac{\|r\|}{r_\oplus}$ where $r \in \mathbb{R}^3$ is the spacecraft position and r_\oplus the radius of the earth, then the form factor is

$$F_{\text{form}} = \begin{cases} \frac{\cos(\zeta)}{H^2} & \zeta < \frac{\pi}{2} - \sin^{-1}\left(\frac{1}{H}\right) \\ F_{\text{form},2} & \frac{\pi}{2} - \sin^{-1}\left(\frac{1}{H}\right) < \zeta < \frac{\pi}{2} + \sin^{-1}\left(\frac{1}{H}\right) \\ 0 & \zeta > \frac{\pi}{2} + \sin^{-1}\left(\frac{1}{H}\right) \end{cases} \quad (19)$$

with

$$\begin{aligned} F_{\text{form},2} &= \frac{1}{2} - \frac{1}{\pi} \sin^{-1}\left(\frac{\sqrt{H^2 - 1}}{H \sin(\zeta)}\right) \\ &\quad + \frac{1}{\pi H^2} \left(\cos(\zeta) \cos^{-1}\left(-\sqrt{H^2 - 1} \cot(\zeta)\right) \right. \\ &\quad \left. - \sqrt{H^2 - 1} \sqrt{1 - H^2 \cos^2(\zeta)} \right). \end{aligned}$$

2.5 Shadow Function

The shadow function gives the occultation of the satellite due to the earth. We use cylindrical shadows as illustrated in Figure 6. The distance between earth and sun is way higher than the difference of their radii and the distance between earth and spacecraft, which is why it is sufficient to assume cylindrical shadows instead of conic ones. We construct an orthonormal basis $\{x, y, z\} \subset \mathbb{R}^3$ with $x = \frac{s}{\|s\|}$ then the shadow coefficient $\nu = \nu(r, s)$ is calculated as

$$\nu = \begin{cases} 1 & \text{if } r^\top x < 0 \wedge \|r^\top y + r^\top z\| < r_\oplus \\ 0 & \text{otherwise} \end{cases}. \quad (20)$$

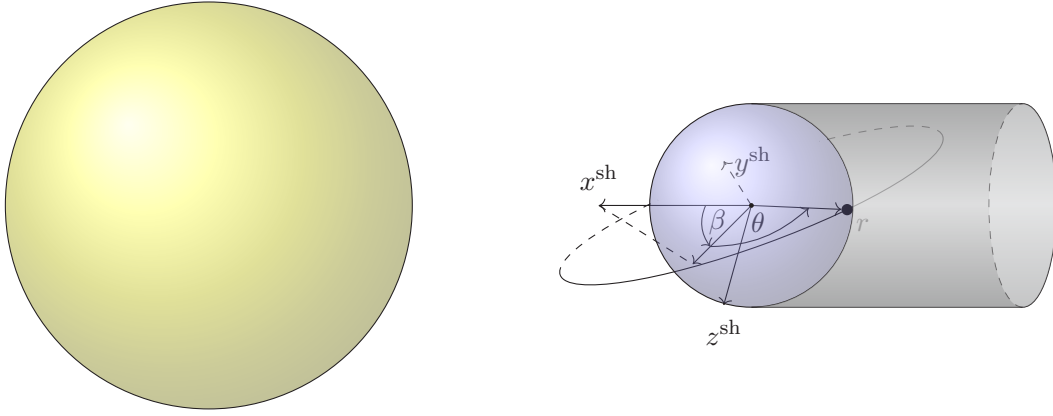


Figure 6. Cylindrical Shadow Model

Note that instead of taking an arbitrary normal basis we can define a coordinate system $^{\text{sh}}$ using the defined solar noon and beta angle via

$$\begin{aligned} T^{\text{sh},\text{I}} &= [x^{\text{sh}} \quad y^{\text{sh}} \quad z^{\text{sh}}]^{\top} \\ &= R_y(-\frac{\pi}{2})R_x(\beta)R_y(\theta)R_y(\pi)T^{\text{o},\text{I}}, \end{aligned}$$

where

$$R_x(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}.$$

We can use this coordinate system to parameterise r as

$$\frac{r}{\|r\|} = \cos(\theta)\cos(\beta)x^{\text{sh}} + \cos(\theta)\sin(\beta)y^{\text{sh}} - \sin(\theta)z^{\text{sh}}.$$

Then Equation (20) reads

$$\nu = \begin{cases} 1 & \text{if } |\theta| > \frac{\pi}{2} \wedge \sqrt{\cos(\theta)^2 \sin(\beta)^2 + \sin(\theta)^2} < \frac{r_{\oplus}}{\|r\|} \\ 0 & \text{otherwise} \end{cases}. \quad (21)$$

Other methods divide the earth's shadow into umbra and penumbra. The shadow coefficient $\nu \in (0, 1)$ in penumbra is then determined by the overlapping of two circular disks. A detailed derivation can be found in (Montenbruck and Gill 2011).

3 Modelica Implementation

The implementation of the Thermal Space library is an extension of the DLR Space Systems library from (M. J. Reiner and Bals 2014) and uses gravity and sun models of the DLR Environment Library (Briese, Klöckner, and M. Reiner 2017). The implemented models are based on the Modelica Standard Library.

3.1 Heat Fluxes Implementation

Each of the solar radiation, albedo radiation, infrared radiation and deep space radiation is implemented. We will discuss only the implementation of the Albedo radiation in detail as all other radiations follow the same implementation concept. The albedo model is shown in Figure 7. The user may provide the material specific solar absorptance parameter α as well as the area of the surface A and the normal of the surface n^{B} in body coordinates. Additionally, the average solar flux constant $G_{\text{s}0}$ and the albedo coefficient ρ_{alb} may be provided. Standard values for these parameters exist, however it is often desired to simulate special hot and cold case scenarios which makes an adaption of these parameters as implemented a desirable feature. The model has two ports, a frame and a heat port connector. As the spacecraft is usually modelled as a rigid body using the Modelica Multi-Body Library (Otter, Elmqvist, and Mattsson 2003), the frame connector has to be connected to the body modelling the spacecraft. Like this the orientation of the frame can be accessed to provide the position r and orientation of the spacecraft T^{B} . Additionally, the outer world model is used to obtain the position of the sun s . Then Equations (8) and (16) are used to determine the solar zenith angle ξ . The orientation of the spacecraft is used to transform the normal vector in body coordinates n^{B} into ECI coordinates n using Equation (1). Then the position of the spacecraft r and the normal of the surface n are used to determine the form factor with Equation (19). Finally the albedo heat flow Q^{alb} is calculated using (4) and fed to the heat port as can be seen in Figure 7. This heat port can then be connected to other sources and sinks of heat to model the thermal dynamics. Instead of using (16), Equation (17) can be used with (9), (12), (13) and (14) to describe the influence of the solar angle. This gives the same results but uses the beta angle β instead of the solar zenith angle ξ which may be easier to parameterise with respect to the satellites orbit as can be seen in (15). The other radiations have

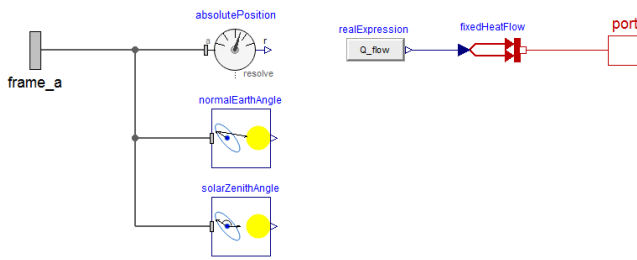


Figure 7. Albedo Model Diagram

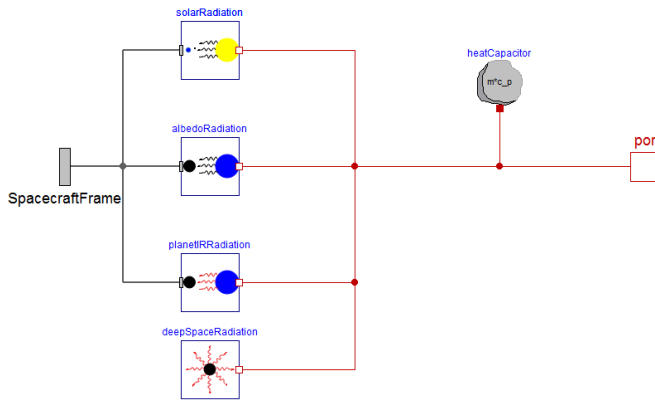


Figure 8. Spacecraft Surface Model Diagram

the same structure but use the Equations (3), (5) and (7), respectively, with the angle defined in (18) and the shadow function (20).

3.2 Thermal Space Components

The thermal model of a spacecraft surface can be seen in Figure 8. The thermal dynamics are described by the differential equation

$$C\dot{T} = Q^{\text{alb}} + Q^{\text{sun}} + Q^{\text{planet}} - \varepsilon A \sigma T^4 + Q^r \quad (22)$$

where C is the thermal capacitance of the surface and Q^r describes all other heat fluxes which are acting on the heat port. This includes foremostly the internal power dissipation of the satellite. The capacitance is implemented as a conditional component. This model offers the opportunity to remove the thermal capacitance if only the steady state calculations are of interest. Additionally, a desired temperature of the surface may be given to obtain the necessary dissipative power which have to be for example produced by heaters to maintain this temperature.

Since many small satellites have the form of a cuboid, a model with six spacecraft surfaces with an infinite resistance between them is implemented. This can be used to simulate the heat evolution at each spacecraft surface as in Section 4. In order to account for the different satellite modes, attitude specific surface configurations are implemented as for e.g. earth pointing mode in which the attitude of the satellite is

fixed. Satellite components are modelled as a thermal capacitor which is connected to a spacecraft surface, usually a radiator. For each of these components the parameters already discussed may be provided to simulate different scenarios of interest.

3.3 Architectures

There are three thermal concepts commonly used for micro- and nano satellites as described in (Baturkin 2005) - autonomous concept, centralized concept and combined concept. Each of these structures is implemented modelling the thermal coupling between each thermal component and the external heat exchange.

4 Example Scenario

In order to show the functionality of the library, the thermal dynamics of a cuboid earth pointing spacecraft are simulated. The cuboid is modelled by six surfaces having the properties of a radiator. The surfaces have the same area $A = 1\text{m}^2$ and thermal properties $\alpha = 0.25$ and $\varepsilon = 0.88$. The spacecraft is in a sun synchronous orbit with an altitude of 600km and 10 : 30h longitude of the ascending node simulated 2018-02-10 at 10 : 00h. The earth's gravitation field is approximated up to the second zonal coefficient (Markley and Crassidis 2014). No dissipative heat is simulated and the parameter are chosen as $G_s = 1361\text{W m}^{-2}$, $\rho_{\text{alb}} = 0.3$ and $T_p = 255\text{K}$. One complete orbit, which takes about $5800\text{s} \approx 97\text{min}$, is simulated. The satellite is earth pointing over the whole orbit, i.e. the spacecraft body axes which are perpendicular to the cuboid surfaces are aligned with the orbit frame.

Figure 9 shows the visualisation of the described scenario. The spacecraft itself is visualised as a simple grey cuboid. The heat flows, the sum of solar, albedo and infra-red radiation, acting on each surface are visualised using head up displays from the Visualization library (Bellmann 2009). It can be seen that all but the zenith direction are influenced by a constant heat flow due to the earth's infra-red radiation. Furthermore, it can be seen that the spacecraft is in the sunlight after approximately 1100s up to 4990s and that the transition between shadow and sunlit is discontinuous. The nadir direction is mainly influenced by the infra-red and albedo irradiation as its view to sun is mostly blocked by the earth. Due to the low solar absorptance of the surface the change of the acting heat flow is comparatively small. The zenith direction however is mostly influenced by the solar radiation. No albedo and infra-red radiation reaches this surface. The surface perpendicular to the orbit plane and in sun direction is foremostly influenced by the solar radiation as well. However, due to the small change of the angle between this surface and the direction to the sun, this heat flow is almost piecewise constant. On the contrary, the surface perpendicular

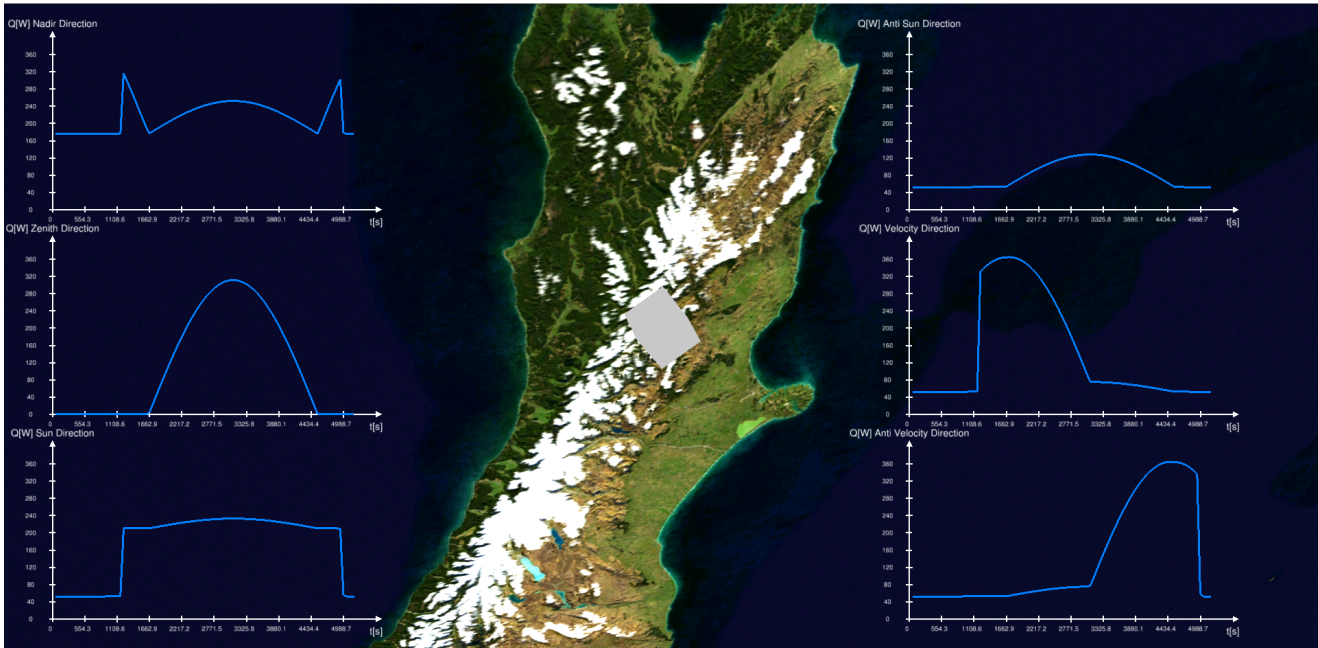


Figure 9. Heat flows acting on the surfaces of a cuboid earth pointing spacecraft

to the orbit plane and in anti sun direction is only influenced by the infra-red and albedo radiation. Due to the small solar absorptance, the albedo radiation influence is comparatively small and this surface has the smallest heat flow changes. The surfaces in velocity and in anti velocity direction are mirrored with respect to the solar noon of the spacecraft. Albedo and infra-red radiation are acting continuously on these surfaces while the influence of the solar radiation can be seen in the sudden discontinuity of the heat flow. All in all, it can be observed that all surfaces are subject to high heat flux changes especially when the spacecraft enters and exits the eclipse. The smallest variation and overall incident heat flux acts on the surface orthogonal to the orbit plane in anti sun direction making it suitable as a surface with radiator.

5 Conclusions

We have presented a Modelica library suitable for the development of a thermal spacecraft model. The main acting environmental and spacecraft heat flows are introduced and their dependence on different angles is given as in the literature. Issues regarding the determination of these angles have been described and novel methods for their calculation are given and discussed. An application example of the proposed library is given to demonstrate the usefulness and flexibility of the Modelica implementation.

References

Baturkin, Volodymyr (2005). “Micro-satellites thermal control—concepts and components”. In: *Acta Astronautica* 56.1-2, pp. 161–170.

- Bellmann, Tobias (2009). “Interactive simulations and advanced visualization with modelica”. In: *Proceedings of the 7th International Modelica Conference; Como; Italy; 20-22 September 2009*. 043. Linköping University Electronic Press, pp. 541–550.
- Briese, Lâle Evrim, Andreas Klöckner, and Matthias Reiner (2017). “The DLR Environment Library for Multi-Disciplinary Aerospace Applications”. In: *Proceedings of the 12th International Modelica Conference*. 132, pp. 929–938.
- Fortescue, Peter, Graham Swinerd, and John Stark (2011). *Spacecraft systems engineering*. John Wiley & Sons.
- Gilmore, David G and Mel Bello (1994). *Satellite thermal control handbook*. Vol. 1. Aerospace Corporation Press EI Segundo, CA.
- Juul, N. H. (1979). “Diffuse Radiation View Factors from Differential Plane Sources to Spheres”. In: *Journal of Heat Transfer* 101.3, p. 558.
- Larson, Wiley J. and James R. Wertz (1991). *Space Mission Analysis and Design*. Springer.
- Lefeng, Sun et al. (2017). “Modeling and Simulation on Environmental and Thermal Control System of Manned Spacecraft”. In: *Proceedings of the 12th International Modelica Conference*. 132. Linköping University Electronic Press, pp. 397–405.
- Markley, F. Landis and John L. Crassidis (2014). *Fundamentals of Spacecraft Attitude Determination and Control*. Springer.
- Meseguer, J, I Pérez-Grande, and A Sanz-Andrés (2012). *Spacecraft Thermal Control*. Woodhead Publishing.

- Montenbruck, Oliver and Eberhard Gill (2011). *Satellite Orbits: Models, Methods and Applications*. Springer.
- Otter, Martin, Hilding Elmqvist, and Sven Erik Mattsson (2003). “The New Modelica Multibody Library”. In: *3rd International Modelica Conference*, pp. 311–330.
- Qian, Jing et al. (2015). “Projection-based reduced-order modeling for spacecraft thermal analysis”. In: *Journal of Spacecraft and Rockets* 52.3, pp. 978–989.
- Reiner, Matthias J and Johann Bals (2014). “Nonlinear inverse models for the control of satellites with flexible structures”. In: *Proceedings of the 10th International Modelica Conference*. 096, pp. 577–587.
- Ruan, Hui, Xiaoguang Hu, and Dan Sun (2017). “Simulation design and implementation of thermal control subsystem for satellite simulator”. In: *12th IEEE Conference on Industrial Electronics and Applications*. IEEE, pp. 1260–1263.
- Tsai, Jih-Run (2004). “Overview of satellite thermal analytical model”. In: *Journal of spacecraft and rockets* 41.1, pp. 120–125.

Exergy Analysis of Thermo-Fluid Energy Conversion Systems in Model-Based Design Environment

Daniel Bender

Institute of System Dynamics and Control, DLR German Aerospace Center, Germany Daniel.Bender@dlr.de

Abstract

Exergy-based analysis has been emerging as a powerful tool for the evaluation of energy intensive systems. Exergy is the maximum theoretical useful work obtainable as the system is brought into complete thermodynamic equilibrium with the thermodynamic environment. Besides the thermodynamic efficiency, both the real thermodynamic value of an energy carrier and the real thermodynamic inefficiencies within a system can be identified. Environmental control systems (ECS) of aircraft as highly interacting systems are an ideal candidate for exergy-based analysis. The design task on architectural level is currently performed using model-based design methods. However, if such systems are evaluated from an exergetic point of view, the analysis is done subsequent of the model-based simulations using rudimentary tools. This work presents a way how exergy-based methods can be integrated into the model-based design environment of Modelica with focus on generic compatibility.

Keywords: exergy analysis, thermo-fluid systems, energy conversion systems, aircraft ECS

1 Introduction

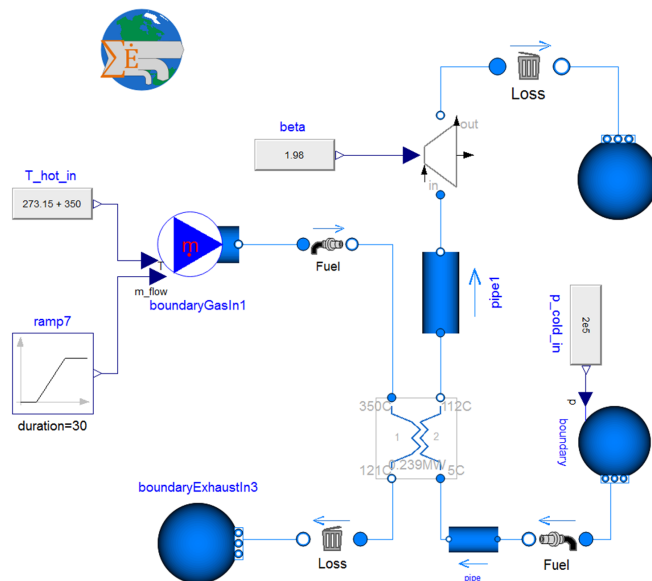


Figure 1. Modelica diagram of power generation cycle partly modeled with components from MSL.

Exergy analysis is a very specific field of evaluation methods for energy conversion processes and usually not well known by the broad audience. A simple example of its evaluation capability is given right here in the beginning to support the understanding of the further work.

Figure 1 shows the Modelica diagram of a simple power generation process. Cold air is preheated within a heat exchanger and then expanded while passing a turbine to produce power. Using exergy-based methods one can give information about the real thermodynamic value of the energy supplied (Fuel, E_{fuel}) to the system, how much is discharged to the environment (Loss, E_{loss}) and wasted due to inefficiencies, the so-called exergy destruction (E_D).

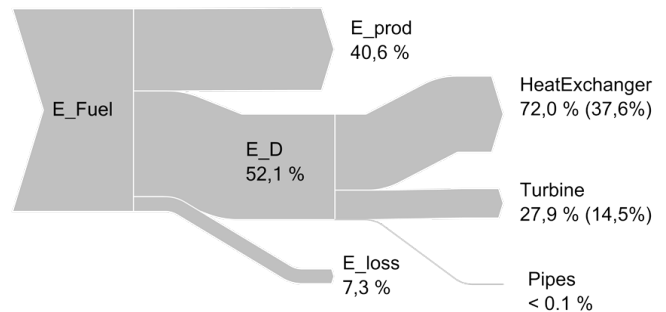


Figure 2. Grassmann diagram showing the results of the exergy analysis for the power generation cycle of Figure 1.

Figure 2 shows a Grassmann diagram (Grassmann, 1950) of the simulation results for the power generation cycle. The analysis shows that 40.6% of the supplied fuel exergy is only used to produce power in the turbine. 52.1% is wasted due to inefficiencies in the components and 7.3% is discharged by the exiting flows leaving the system. Further the results identify the heat exchanger as the main source of inefficiencies. It is responsible for 72.0% of the exergy destruction, which is 37.6% of the supplied fuel exergy. The turbine causes less with about 27.9% and 14.5%, respectively. The losses within the pipes can be neglected. The outcomes of the exergy analysis give not only information about the total inefficiency of the process, but also about the impact of the single components.

This example shall convey an idea of exergy-based methods and their significance. The focus of this work is to present the integration of exergy-based methods into the

model-based design environment of Modelica. Within this section, state of the art tools for the modeling of thermo-fluid systems and evaluation of such are discussed. Then, a brief introduction to the methodology of exergy analysis is given and their application within this work. The next section presents the integration of the exergy-based methods into the model-based environment and two examples are shown. Finally, the paper is concluded with some remarks.

Model-based design methods have become well established methods for numerical simulations of complex thermo-fluid systems. This comprises the simulation of single components and large scale systems with large time scales. Much effort have been put into the modeling of energy intensive systems for different fields of applications. Ranging from basic thermo-fluid modeling (Elmqvist et al., 2003) to automotive refrigeration systems (Limperich et al., 2005), large buildings simulation (Wetter et al., 2014) and environmental control systems of aircraft (Sielemann et al., 2007).

The modeling and simulation process as part of a design process usually comes along with subsequent evaluation and optimization tasks of the system of interest. Optimization capabilities for different kinds of applications are provided by specific libraries (Pfeiffer, 2012; Bender, 2016). The evaluation of energy conversion systems is performed in most cases using evaluation criteria based on the first law of thermodynamics. Energy balances are formulated and the efficiency of a conversion process is measured by comparing the supplied energy with the desired output of the process. The difference of the supplied energy and output represents the waste energy, in other words the losses of the process. Most component models of energy conversion processes, such as turbo components or heat exchangers are either described by thermodynamic efficiencies based on the first law or are equipped with such. Hence, an additional library or tool is not necessary. Unfortunately, this does not apply for analysis that asks for extended questions such as the following topics. Environmental or economical issues are usually not mandatory for the simulation of the energy conversion process and are influenced by additional factors such as costs and system specific aspects linked to their field of application. (Wischhusen et al., 2003) present the simulation and optimization of a complex industrial energy system with respect to economic benefits. The analysis was performed using an applied simulation tool based on Modelica. A physical domain independent library was developed by (Zimmer and Schlabe, 2012). Economic models are provided for the implementation into Modelica and allow energy management tasks.

Exergy analysis can be seen as an extended thermodynamic analysis that requires a different view of the energy conversion process. The exergy equations need only basic math and are usually performed subsequent of the simulation process. When it comes to exergy analysis of dynamic systems, a subsequent analysis needs more effort as the

dataset increases significantly. (Sanghi et al., 2014) developed a Modelica-based tool to provide a dynamic exergy analysis for buildings simulation. The library followed a similar approach as it is presented in this paper. Unfortunately, the library is not published and the formulation of the exergy equations are not documented. Therefore, no statement is possible if it meets the requirements for the application to aircraft environmental control systems. This circumstance leads directly to the motivation of the work presented here.

The question of how exergy analysis can be integrated into the model-based design environment developed during the endeavors for a comprehensive formulation of exergy-based methods for aircraft environmental control systems.

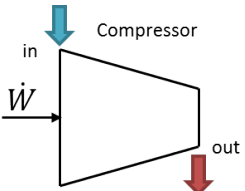
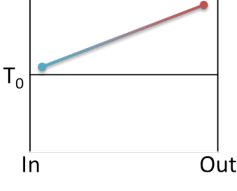
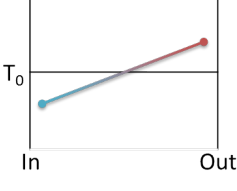
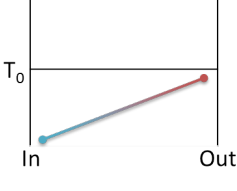
The definition of exergy balances highly depends on the actual thermodynamic states of the working fluid and the conditions of the reference state. Aircraft operate among highly varying environmental conditions that impact temperature, pressure and humidity. The same applies to aircraft ECS. With changing ambient environment, the ECS operates on different operation points. Depending on the definition of the reference environment for the exergy analysis, any possible situation must be considered for the integration of exergy-based methods. The impact of the reference environment on the definition of the exergy balances is consolidated later in this paper.

Based on the demands derived from the exergy-based methods and the ongoing development of aircraft ECS using model-based design, a library is presented with the aim to allow exergy-based analysis of aircraft environmental control systems. The application of this library shall not be limited to aircraft applications and library structures currently used for the modeling and simulation of aircraft ECS (Sielemann et al., 2007; Zimmer et al., 2018). Both libraries have been developed in collaboration with an industry partner and are not publicly available at their current development stage.

2 Exergy analysis

Energy balances consider the quantity of energy, but neglects to express the quality of energy. The real thermodynamic value, i.e. its quality, of an energy source gives information about its potential to cause a change in a useful way. Kinetic, potential, mechanical and electric energy can be transformed in an ideal process to any other form of energy. The quality of thermal and chemical energy, however, depends on the state of the energy carrier (temperature, pressure and chemical composition) with respect to the environment. In thermodynamics, *exergy* characterizes the quality of a given quantity of energy. Using an energy balance in combination with the second law of thermodynamics, both the thermodynamic true value of an energy carrier, and the real thermodynamic inefficiencies can be determined. This is possible for a single process and on system level. Within the system boundaries, the occurring

Table 1. Definition of exergy of fuel and exergy of product.

| Component schematic | Operation Conditions | Exergy Rates |
|---|--|--|
|  |  <p style="text-align: center;">$T_{in} \geq T_0$</p> | $\dot{E}_{Fuel} = \dot{W} + \dot{E}_{in}^{Ch} - \dot{E}_{out}^{Ch}$ $\dot{E}_{Prod} = \dot{E}_{out}^{Ph} - \dot{E}_{in}^{Ph}$ |
| |  <p style="text-align: center;">$T_{in} \leq T_0 \leq T_{out}$</p> | $\dot{E}_{Fuel} = \dot{W} + \dot{E}_{in}^T + \dot{E}_{in}^{Ch} - \dot{E}_{out}^{Ch}$ $\dot{E}_{Prod} = \dot{E}_{out}^{Ph} - \dot{E}_{in}^M$ |
| |  <p style="text-align: center;">$T_{out} \leq T_0$</p> | $\dot{E}_{Fuel} = \dot{W} + \dot{E}_{in}^T - \dot{E}_{out}^T + \dot{E}_{in}^{Ch} - \dot{E}_{out}^{Ch}$ $\dot{E}_{Prod} = \dot{E}_{out}^M - \dot{E}_{in}^M$ |

real thermodynamic inefficiencies are exergy destruction and exergy transfers out of the system are regarded as exergy losses (Bejan et al., 1996).

(Tsatsaronis, 2007) specified the definition that the exergy of a thermodynamic system is the maximum theoretical useful work obtainable as the system is brought into complete thermodynamic equilibrium with the thermodynamic environment while the system interacts with this environment only.

The total flow exergy of a fluid stream i is expressed as:

$$\dot{E}_{t,i} = \dot{m} \cdot [h_i - h_0 - T_0 \cdot (s_i - s_0)] \quad (1)$$

where \dot{m}_i is the mass flow, and h and s represent the specific enthalpy and specific entropy of the fluid stream i and reference environment 0.

The exergy flow balance for the k – th component is defined by:

$$\dot{E}_{F,k} = \dot{E}_{P,k} + \dot{E}_{D,k} \quad (2)$$

where subscripts F, P and D represent the fuel exergy, product exergy and destroyed exergy of the k – th component. (Lazzaretto and Tsatsaronis, 2006)

The balance for the total system can be written as:

$$\dot{E}_{F,tot} = \dot{E}_{P,tot} + \sum_k \dot{E}_{D,k} + \dot{E}_{L,tot} \quad (3)$$

with tot representing the total amount of the overall system.

The exergetic efficiency ϵ_k of the k – th component is defined by the following expression:

$$\epsilon_k = \frac{\dot{E}_{P,k}}{\dot{E}_{F,k}} = 1 - \frac{\dot{E}_{D,k}}{\dot{E}_{F,k}} \quad (4)$$

The rate of exergy destroyed related to the exergy of total fuel is expressed by the exergy destruction ratio:

$$y_{D,k} = \frac{\dot{E}_{D,k}}{\dot{E}_{F,tot}} \quad (5)$$

The approach of defining the exergy balance using fuel and product exergy instead of entering and exiting energy flows is explained in detail by (Lazzaretto and Tsatsaronis, 2006) and (Tsatsaronis and Morosuk, 2013). The product exergy \dot{E}_P represents the desired result (expressed in terms of exergy) generated by the system being considered. The fuel exergy \dot{E}_F represents the general resources in terms of exergy that are expended to provide the product exergy. The fuel and product parts of a component are determined by considering the physical exergy \dot{E}^{Ph} , i.e. thermal \dot{E}^T and mechanical \dot{E}^M parts, and chemical exergy \dot{E}^{Ch} of each stream. The definition of the balances is done for each component differently depending on its aim, i.e. the balances for a heat exchanger differ from the balances for an expansion device. Additionally

to the component's aim, the reference environment has to be taken into account. (Bender, 2017) gives a comprehensive overview for typical energy conversion processes of an aircraft environmental control system. Table 1 shows

Table 2. Component categories and number of operation conditions considered for the definition of exergy rates.

| <i>Component categories</i> | <i># of Cases</i> |
|-----------------------------|-------------------|
| Heat exchanger | 6 |
| Compressor | 3 |
| Turbine | 3 |
| Water extraction | 1 |
| Water injection | 3 |
| Mixer | 1 |
| Flow resistance | 1 |

exemplary the definition of the exergy rates for a turbo driven compressor or fan. During the compression, external power is supplied to the process and increases the pressure and concurrently the temperature of the entering fluid stream. Applying now the methodology of fuel and product exergy, the supplied power belongs to the fuel and the pressure increase counts as product exergy. The chemical exergy does not change among the process for most cases. It needs to be considered for a comprehensive approach because it could change in the case of having a saturated moist air flow and evaporation takes place during the compression. The details of the behavior of the chemical exergy are not discussed here as they would exceed the scope of this work. However, the thermal exergy that depends on the temperature changes differently from the pressure. It reaches its minimum of 0 at the reference temperature and increases with increasing deviation from T_0 . Depending on the operation condition, three exergy balances can be identified. The operation conditions define in this context at which temperature the compressor operates with respect to the reference environment, in particular the reference temperature T_0 . The first case considers the operation above the reference environment. The second case includes the crossing of T_0 . During the third case, the temperature at the outlet remains below T_0 .

3 Integration into model-based environment

The analysis of thermo-fluid energy conversion systems using exergy-based methods is usually performed in two steps. The system is simulated in a first step using a simulation environment, such as Ansys (Ansys, 2018), Aspen Plus (aspentech, 2018) or Modelica. The produced thermodynamic data is then used to do the exergy analysis in a subsequent step. This requires a data transfer to another calculation software (e.g. EES (F-Chart, 2018) or MS Excel). If the analysis is limited to one or a few operation points, the amount of data remains manageable. But if the exergy analysis shall be performed for several operation

points or include dynamic behavior of a system, the produced data exceed soon a practicable amount.

Modelica is already used for the application of large thermo-fluid systems. Evaluation criteria based on the first law of thermodynamics such as energetic efficiencies are provided in most components that describe energy conversion processes. The advantage of the model-based approach that system models can be built from scratch or modified using available libraries within a short time shall now be extended with the capability of exergetic-based methods. The exergy analysis is a subsequent calculations step and does not impact the system behavior.

3.1 Requirements

In order to achieve a solution as generic as possible some requirements need to be formulated derived from both, the exergy-based methods and the model-based environment:

Exergy-based Methods

- Retrieve thermodynamic state of all energy streams entering and exiting a component
- Identify the aim the component's energy conversion process
- Select the appropriate exergy balance of fuel and product exergy rates depending of the operation condition and reference environment
- Allow a user defined exergy analysis on system level using the component's based analysis
- Centralized propagation of reference environment on system level among all components
- Media models must provide appropriate functions to calculate further thermodynamic data

Model-based environment

- Generic approach for easy integration into any thermo-fluid library
- Compliant with Modelica Standard Library (Usage of MSL Media models and MSL Fluid connectors)
- Minor impact on numerical computation

The best way to put these requirements into action would require the screening of the modeled system architecture. The structure could be gathered to identify the components' aims and link them with their appropriate fuel and product definition. This procedure can be seen as a preceding step before the simulation starts. Unfortunately, Modelica at its recent stage of development used for this work (The Modelica Association, 2013) does not provide sufficient practical sequential capabilities. Therefore the integration needs to be realized in a different way.

Listing 1. Modelica code of exergy implementation in sensor models: Calculation of exergy flows.

```

// Exergy implementation
// -----
(e_t_in,e_therm_in,e_mech_in,e_chem_in) = Functions.exergyFlow_MoistAir(
    airMediumA.state, state_ref);
(e_t_out,e_therm_out,e_mech_out,e_chem_out) = Functions.exergyFlow_MoistAir(
    airMediumB.state, state_ref);
E_t_in = e_t_in * airMediumA.X[2] * m_flow;
E_t_out = e_t_out * airMediumB.X[2] * m_flow;

E_therm_in = e_therm_in * airMediumA.X[2] * m_flow;
E_therm_out = e_therm_out * airMediumB.X[2] * m_flow;

E_mech_in = e_mech_in * airMediumA.X[2] * m_flow;
E_mech_out = e_mech_out * airMediumB.X[2] * m_flow;

E_chem_in = e_chem_in * airMediumA.X[2] * m_flow;
E_chem_out = e_chem_out * airMediumB.X[2] * m_flow;

```

3.2 Component level

The exergetic analysis on component level includes the exergy balance of Equation (2) and the calculation of the exergetic efficiency with Equation (4). Table 1 gives an example of how the fuel and product exergy is defined for a compressor. To calculate the appropriate exergy flows, the thermodynamic data at the component's connectors must be provided. As mentioned before, it is hardly manageable to catch the data from outside the component without huge effort for the user. Therefore the approach of a sensor model was chosen to integrate the exergy equations to the component. To keep the usability as simple as possible, the sensor model must be dropped to the component model and *linked* to the component's connectors and some other variables. Figure 3 shows how the *StaticPipe* model looks after the exergy sensor has been integrated on component level. The sensor model is then *linked* with the component by adding standardized code to the component model to supply the reference environment and the *WorldEx* model. The following listing shows an example of an exergy sensor integrated to a compressor model:

```

// Reference environment
// -----
SIunits.Temperature T_ref =
    worldEx.T_ref
"Reference Temperature for Exergyflow";
SIunits.Pressure p_ref = worldEx.p_ref
"Reference Pressure for Exergyflow";
SIunits.MassFraction X_ref[:] =
    worldEx.X_ref;
outer ExergyLibrary.World worldEx;

//*****Sensors*****
Sensors.Air.ExergySensor_twoPort_turboCmp
    exergySensor_twoPort (
    airMediumA (state=AirMedium.setState_phX (
    portA.p,

```

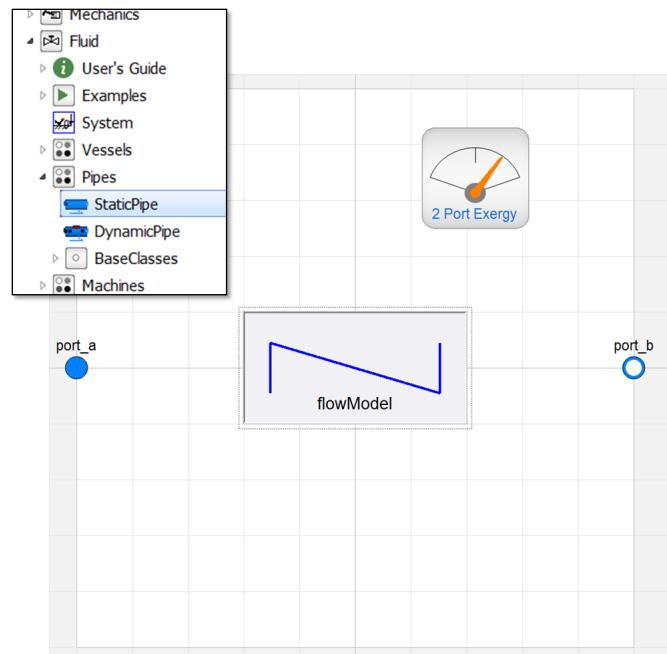


Figure 3. Modelica diagram of *StaticPipe* model with exergy sensor integrated.

```

    portA.h,
    portA.Xi),
    redeclare package AirMedium
    = AirMedium),
    airMediumB (state=AirMedium.setState_phX (
    portB.p,
    portB.h,
    portB.Xi),
    redeclare package AirMedium
    = AirMedium),
    m_flow=m_flow,
    power=power,
    T_ref = T_ref,

```

Listing 2. Modelica code of exergy implementation in sensor models: Definition of fuel and product balances.

```
// Calculation of fuel and product Exergy
if m_flow <= 0 then
  E_fuel = 0;
  E_prod = 0;
  case_T = 0;
else
  if airMediumA.T > state_ref.T then
    case_T = 1;
    E_fuel = abs(power) + E_chem_in - E_chem_out;
    E_prod = E_therm_out - E_therm_in + E_mech_out - E_mech_in;
  elseif airMediumB.T >= state_ref.T and airMediumA.T <= state_ref.T then
    case_T = 2;
    E_fuel = abs(power) + E_therm_in + E_chem_in - E_chem_out;
    E_prod = E_therm_out + (E_mech_out - E_mech_in);
  elseif airMediumB.T < state_ref.T then
    case_T = 3;
    E_fuel = abs(power) + (E_therm_in - E_therm_out) + E_chem_in - E_chem_out;
    E_prod = E_mech_out - E_mech_in;
  else
    case_T = 100;
    E_fuel = 0;
    E_prod = 0;
  end if;
end if;
E_D = E_fuel - E_prod;
exergy_eff = E_prod / max(eps, E_fuel);
```

```
p_ref = p_ref,
X_ref = X_ref);
```

Besides the thermodynamic state at the connectors, the flow medium, mass flow, reference environment and in this particular case of the compressor, the power is transferred. The propagation of the thermodynamic state enables to be independent from the type of connectors used for the system modeling. The same applies for the flow medium. Here, the same medium model that is used for the component is propagated to the exergy sensor. As long as the medium model provides basic equations similar to the MSL, any medium model can be used. All calculations for the exergy flows and the definitions of the exergy rates depending on the operation condition (Table 1) happen within the sensor model. First the specific exergy at the inlet and outlet of the component are calculated. Function models provide the algorithms to determine the specific exergy split into its thermal, mechanical and chemical parts. Listing 1 shows how the function call works. The thermodynamic states at the connector and of the reference environment are committed to the function and the specific exergy values are returned. Following the exergy flows are calculated depending on the dry air and mass flow. The exergy flows of thermal (\dot{E}^T), mechanical (\dot{E}^M), and chemical (\dot{E}^{Ch}) parts, and the exergy of work (\dot{W}) are then used to formulate the fuel and product balances. The Modelica code of the compressor exergy sen-

sor for the exergy rates of Table 1 is presented in Listing 2. It must be mentioned that the thermal and mechanical exergy flows combined are regarded as physical exergy flow: $\dot{E}^T + \dot{E}^M = \dot{E}^{Ph}$. This approach allows the calculation of the exergy destruction within each component. The user integrates the sensor to the component level and does not have to care about the formulation of the exergy equations. The recent exergy library contains a sensor model for each category listed in Table 2.

3.3 System level

A `WorldEx` model controls the exergy analysis on system level. It is responsible for the definition of the reference environment and collects the exergetic information for the exergy balance on system level. The reference environment can be either defined with fixed values or it can be linked to an environment model with variable conditions.

The exergy balance for the total system is formulated by Equation (3). The exergy flows for the total fuel, product and losses must be defined by the user as they depend on the system architecture. For each of the exergy flows, sensor models are available to integrate them to fluid flows. Similar to mass flow sensors, they must be implemented within the appropriate fluid stream. The exergy of the stream is calculated without any impact on the flow. In some cases it is not possible to use fluid sensors to catch fuel, product or loss exergy of the system, e.g. pure power

that is supplied to or extracted from the system. For such cases the GUI of the world model provides input boxes for each part of the exergy balance where the variable names can be entered. A mixed usage of sensors and entered variable names is possible as all parts are captured and summed up automatically.

The capability of automatically collecting the exergy data for the system balance requires the identification of all implemented exergy objects. Unfortunately, during runtime, Modelica does not provide such capabilities to screen a system model for its data structure. With the help of the UID Library (Hellerer and Buse, 2017) all exergy sensors are equipped with a unique identifier which can be identified on system level and used for the system balance. The following listing shows the implementation of the unique identifier to the exergy sensor for the implementation to the components:

```
UID.UniqueID uniqueID(group="exergy") a;
parameter String instanceName =
  getInstanceName();
equation
  worldEx.E_D[uniqueID.uid+1] = E_D;
  worldEx.instanceName[uniqueID.uid+1]
    = instanceName;
```

The UniqueID model is added to the sensor model and assigned with a group. A unique integer value uid is then provided within the group. In this way, it is possible to propagate the exergy destruction calculated within each sensor and the instance name of the component to the WorldEx model. The uid value starts at 0 and is in the range $[0..total[$. The World model contains a GroupTotal object that provides the total number of values assigned within a certain group:

```
UID.GroupTotal groupTotal(group="
  exergy") a;
Modelica.SIunits.Power E_D[groupTotal.
  total];
Modelica.SIunits.Power E_D_total = sum
  (E_D) + E_D_user;
String instanceName[groupTotal.total];
```

This allows to summarize the single values of the exergy destruction to the total destroyed exergy within the considered system. The collection of the fuel, product and loss exergy flows works in a similar way using additional GroupTotal objects having different group names assigned.

The WorldEx model provides the list of the collected instance names of the components that have an exergy sensor implemented in the *Dymola Message Window* and writes them into an extra text file.

3.4 Library structure

The structure of the library is shown in Figure 4. The WorldEx model is available on the top level. Besides the User's Guide and Examples package there are three packages on the top layer. The Functions package includes all functions that are necessary to calculate the exergy flows

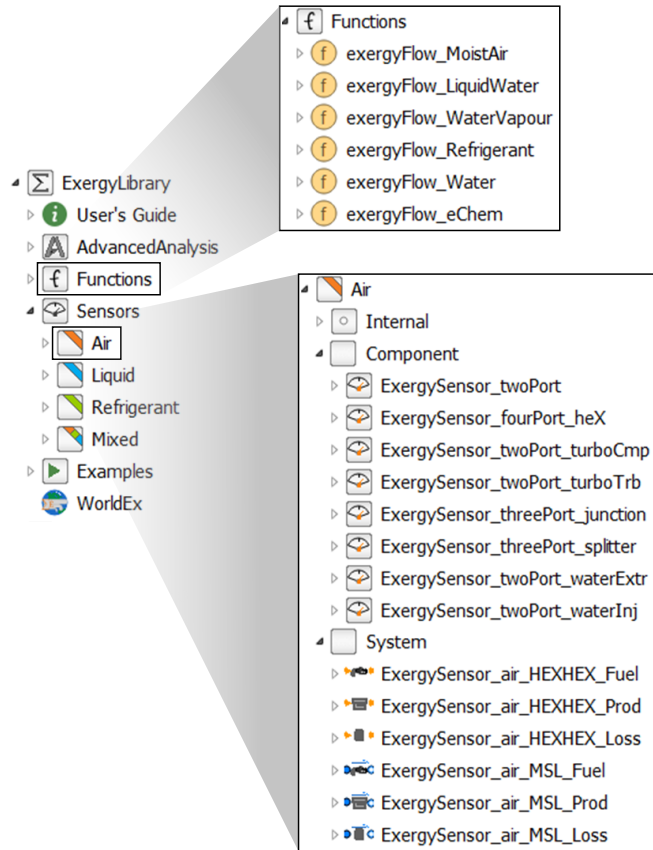


Figure 4. Library structure of the exergy library.

and called from the exergy sensor models (Listing 1). The Sensors package contains all sensor models. It is organized according the fluid medium. Currently there are sensors available for air, liquid, refrigerant, and mixed media. Typical applications for mixed media sensors are for example heat exchangers with two media. All sensors are valid for unidirectional flow. The examples package aims at the understanding of how the sensor models are applied. Finally, there is an additional package for advanced analysis applications that cover aircraft specific calculations such as fuel weight penalties and the calculation of the flow exergy of aviation jet fuel.

The sensor package is further organized in component and system level. The component package contains the sensors for the different component types. Currently eight processes are covered: flow resistances and valves, heat exchangers, power demanding and producing turbo components, junctions such as splitter and mixer, and water extraction and injection. The System package provides sensor models to perform exergy analysis on system level according Equation 3. The sensor models equal mass flow sensors and need to be integrated in between a flow stream. These sensors must be compatible with the infrastructure, i.e. have the same connectors, of the used library. For the moment there are sensors provided that are compatible with the Modelica standard library and a new approach of thermo-fluid modeling that is presented

in detail by (Zimmer et al., 2018).

4 Examples

Within this section, two examples will show how the exergy library is applied to thermo-fluid systems.

Modelica Standard Library

The simple power generation cycle with moist air as working fluid has already introduced this work in the first section. Cold air is preheated within a heat exchanger and then expanded while passing a turbine to produce power. Figure 1 shows the diagram layer of the Modelica model. On the top layer the `WorldEx` model and the exergy sensors for the system balance, Loss and Fuel can be seen. The produced power of the turbine states the product of the total exergy balance. The integration of the power value to the exergy balance works by writing the variable name to the GUI of the `WorldEx`, Figure 5. The exergy sensors

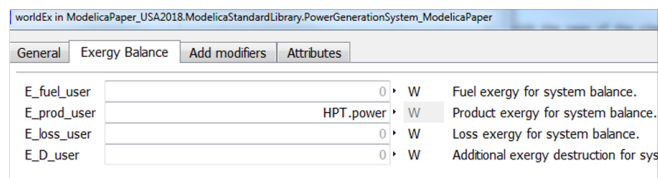


Figure 5. Extract of the `WorldEx` GUI showing the assignment of the turbine power to the product exergy of the system balance.

for the components are integrated on component level. An example is shown in Figure 3. Once the exergy sensors have been implemented, the model can be simulated in the usual way.

In the beginning of the simulation a text file "ExComponentNames_*modelName*.txt", where *modelName* is replaced with the name of the simulated model, is created. It lists all components of the simulated model with exergy sensors inside (Figure 6) and a list of used exergy sensors for the system balance, i.e. fuel, product and loss. In this example there are four components, the turbine (HPT), the heat exchanger (hx_HP) and two pipes, listed with their names and paths to the exergy sensor and two fuel and loss exergy sensors. The path to the exergy sensors show that the naming of the sensors is the same independent of the exergy sensor model. A default name is set for all exergy sensor models. Taking an example from Figure 4, usually when the "ExergySensor_twoPort" model is dropped into a pipe model, the name of the sensor model would automatically be set to "exergySensor_twoPort". To simplify an optional post processing of the results by using the generated *.mat file, the exergy sensor name was set to "exergySensor" per default for all models. For the exergy analysis the type of exergy sensor is not important any more as the appropriate balances are included already. The power generation cycle model was simulated for 50s. It starts at steady state condition and then the mass flow of the hot air supply is reduced between 10s and 40s. The plots of the results are shown in Figure 7. The model is

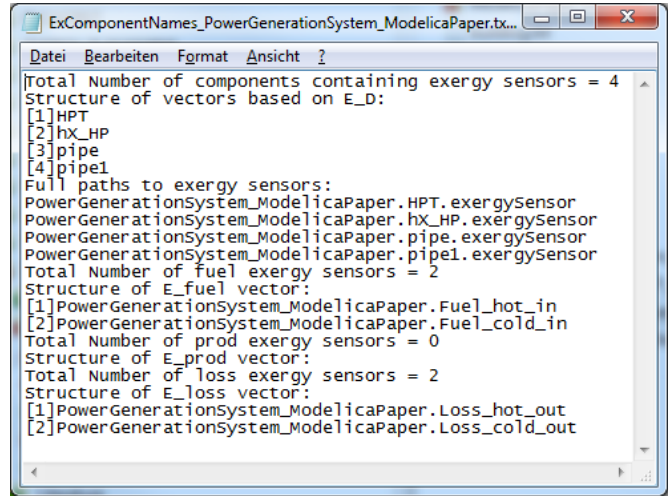


Figure 6. Generated text file with list of components equipped with exergy sensor and list of system exergy sensor models for fuel, product and losses.

just an exemplary application and hence the quality of the results is not further discussed here. But it can be captured that the exergetic values on system level are stored and available within the `WorldEx` model. The results of the fuel and product balances on component level can be found in the variable browser at the paths listed in Figure 6.

Directed thermo-fluid flows using HEXHEX

In the beginning of this paper the requirement of generic and library independent compatibility was emphasized. To show this capability of the exergy library, the second example is modeled using a new approach for robust modeling of directed thermo-fluid flows. This methodology has recently been developed at our institute (Zimmer et al., 2018) with the aim to provide robust modeling for complex networks such as aircraft environmental control systems. The example architecture is much more complex than the example modeled with the MSL, Figure 1. Here the exergy library shows its real advantage for the analysis of large systems.

Figure 9 shows the diagram layer of the second example, an electric driven vapor cycle pack (eVCP) architecture. The architecture is derived from a patent publication (Golle et al., 2016). Unlike the original architecture, the vapor cycle was simplified. The original vapor cycle has an additional evaporator flow through with recirculated air from the cabin. Unlike conventional bleed air driven air cycle packs (Bender, 2017), unconditioned outside air instead of bleed air from the engine enters the eVCP. The cold and low pressure air is compressed in a first stage before it passes the primary heat exchanger (PHX). A second compressor further raises the pressure and temperature before entering the reheater. The main heat exchanger, mounted in the ram air channel, is passed before the evaporator cools the air. In case the saturation temperature is exceeded, a water separator extracts the condensate and

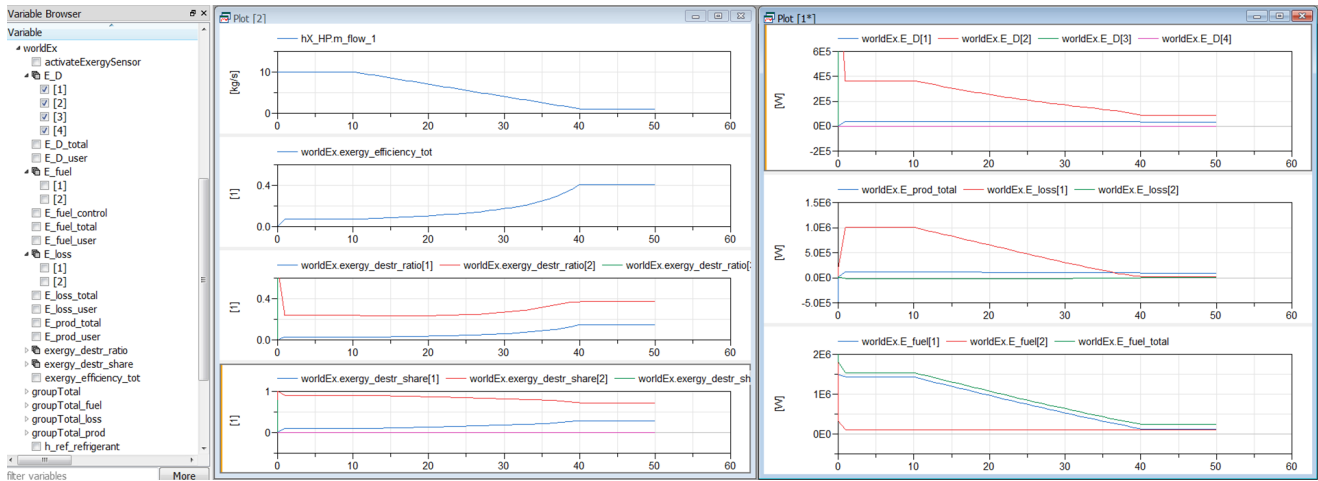


Figure 7. Simulation results of the power generation cycle showing exergy values plotted.

leads it to the water injector located in the the ram air channel upstream the vapor cycle condenser. Before the conditioned fresh air is expanded in the turbine, it is reheated in the reheater. The discharged fresh air meets the recirculated cabin air in the mixing unit downstream the pack. The ram air channel functions as heat sink and is feed with air from the outside. During ground operations, the air flow is provided by a fan located near the ram air outlet. Contrary to a bleed air driven pack which is autonomously driven, all turbo-machines within the eVCP are electrically powered. Within the electric driven vapor cycle pack model, a total number of 30 components with exergy sensors was identified by the exergy library. Figure 8 shows an extract of the text file. Although there are 30 components listed, the exergy sensor was not integrated 30 times manually to the component models. The work has to be done once for each different component class and can then be reused without limitations. The simulation results are available in the variable browser, and stored in the *.mat file, as presented for the MSL example in Figure 7.

5 Conclusion

Risen from the idea to do exergy analysis for aircraft environmental control systems within model-based design, a solution had to be found how to integrate exergy-based methods into the model-based design environment of Modelica. With this motivation, the presented exergy library was developed. The exergy analysis treats every component different, depending on its aim and particular energy conversion process. The derived requirements for such a library coming from both, the exergetic and model-based design, parties resulted in the presented work. An exergy sensor model is implemented into the component model and linked to the entering and exiting energy flows. The sensor then does all calculations for the component level exergy analysis and propagates its instance name and exergy destruction to a WorldEx model. Concurrently, the WorldEx model provides the reference environment for

```

ExComponentNames_eVCP_referenceArchitecture_VaC_TAXI_RAC.txt - Editor
Datei Bearbeiten Format Ansicht ?
Total Number of components containing exergy sensors = 30
Structure of vectors based on E_D:
[1]MXH
[2]Evaporator
[3]Reheater
[4]waterSeparator
[5]waterInjector
[6]fanRamAir
[7]BaseCompressor
[8]ACM_turbine
[9]PHX
[10]ACM_compressor
[11]Condenser
[12]VaCsTurboCompressor
[13]VaCsValve
[14]junction2
[15]junction3
[16]FlowModelAirARamAir4
[17]FlowModelAirARamAir1
[18]junction1
[19]Venturi_ACM_Compressor
[20]Ozone_Converter
[21]Pack_Venturi
[22]waterExtractorAirFlowResistance
[23]AltitudevalveDuctAirFlowResistance
[24]Diffuser
[25]divPlenum
[26]junction4
[27]Base_Compressor_Check_Valve1
[28]junction5
[29]junction6
[30]Base_Compressor_Check_Valve
Full paths to exergy sensors:
eVCP_referenceArchitecture_VaC_TAXI_RAC.MXH.exergySensor
eVCP_referenceArchitecture_VaC_TAXI_RAC.Evaporator.exergys
eVCP_referenceArchitecture_VaC_TAXI_RAC.Reheater.exergySer
eVCP_referenceArchitecture_VaC_TAXI_RAC.waterSeparator.exe
eVCP_referenceArchitecture_VaC_TAXI_RAC.waterInjector.exer
eVCP_referenceArchitecture_VaC_TAXI_RAC.fanRamAir.exergysE

```

Figure 8. Generated text file of eVCP architecture with list of components equipped with exergy sensor and list of system exergy sensor models for fuel, product and losses.

all exergy sensors. The exergy sensor model can be applied independently from the thermo-fluid modeling approach. This was achieved by linking the sensor model not by connecting any fluid connectors but by propagating the thermodynamic states of the entering and exiting fluid flows. The exergy balance on system level can be performed by using additional exergy sensors that work similar to mass flow sensors and need to be connected within the appropriate fluid stream. Unfortunately, these models have to be compatible with the fluid library, i.e. have

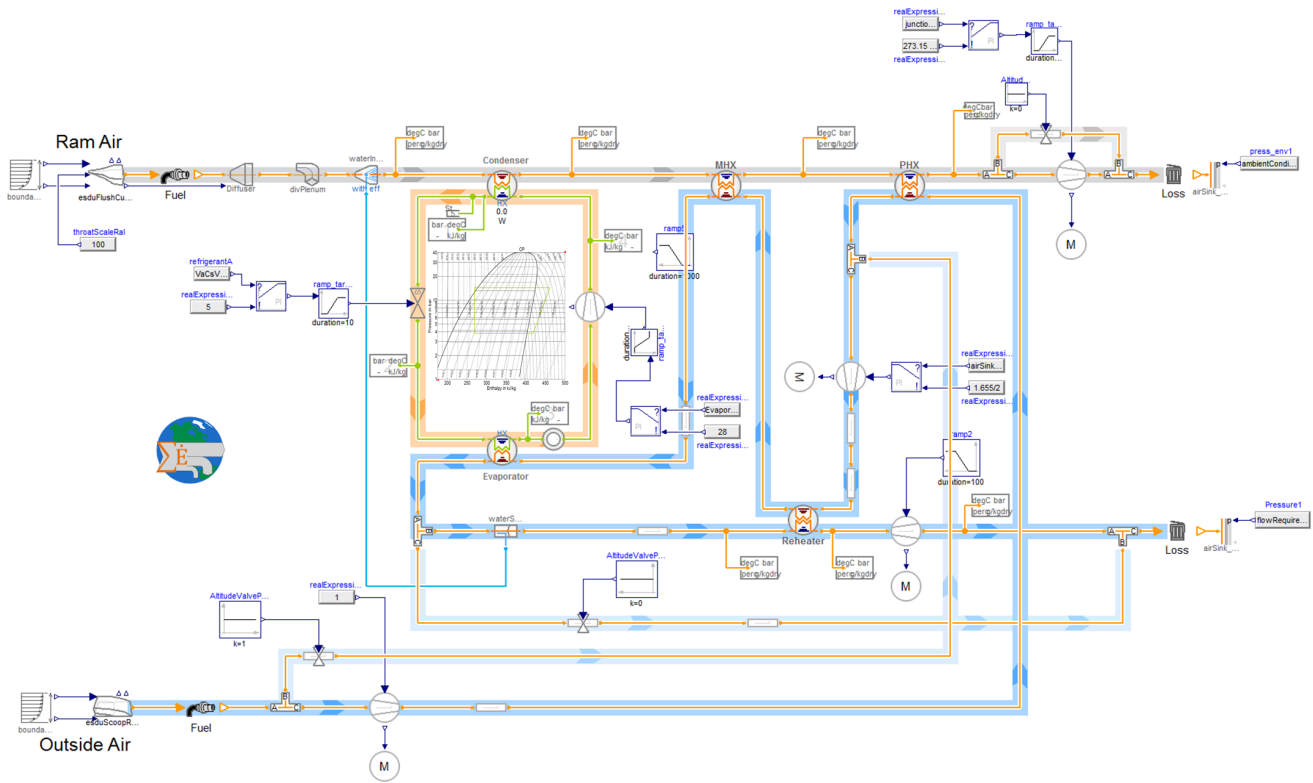


Figure 9. Modelica diagram of electric driven vapor cycle pack architecture modeled using the HEXHEX library.

the same or compatible connectors. The current version of the exergy library provides sensors that can be used with the Modelica Standard Library and the approach of HEXHEX (Zimmer et al., 2018). Not accessible energy streams that should be included to the system balance can be added by writing their variable names into the GUI of the WorldEx model. Here a box is provided for exergy destruction, fuel, product and loss exergy. Additional evaluation numbers such as exergy destruction rate (component and system level) or the ratio of $E_{D,k}/E_{D,tot}$ are provided by the WorldEx model.

With the experience gained during the development of this exergy library, one can say that there is no universal solution as exergy-based methods by their nature, have the individual aspect on the component's aim. But there usually is a limited number of energy conversion processes and components classes. The work for integrating the exergy sensors to the components, needs to be done once and one can benefit from it any time after. The structure of the library allows an easy extension for new energy conversion processes. The advantage of this concept is that the exergy analysis is totally detached from the component development and behavior and allows an as individual use as possible for the modeler.

Remarks

The current version of the exergy library uses mainly the media models of the Modelica Standard Library. For moist air, water and single gases that can be treated as ideal gases, the provided models of the MSL might be suf-

ficient. The second example architecture presented here, has a vapor cycle included that runs with refrigerant. An additional media library was used. The linchpin of the exergy analysis is the correct calculation of the thermodynamic properties. The current thermodynamic state record of the MSL or most other medium models does supply only basic properties which are not sufficient for an exergy analysis (i.e. specific entropy). Therefore, an extended thermodynamic state record was created in the exergy library by using the appropriate equations from the medium models. To further ensure a generic applicability of the exergy library for different kinds of fluids, it is recommended to provide standardized formulation and naming of equations for the thermodynamic properties. Unfortunately, this is not the case for all medium models.

References

- Ansys. Ansys, 2018. URL <https://www.ansys.com/>.
- aspen tech. Aspen plus, 2018. URL <https://www.aspentech.com/products/engineering/aspen-plus>.
- Adrian Bejan, George Tsatsaronis, and Michael J. Moran. *Thermal design and optimization*. A Wiley-Interscience publication. Wiley, New York, 1996. ISBN 978-0-471-58467-4. URL <http://www.loc.gov/catdir/description/wiley032/95012071.html>.
- Daniel Bender. Desa - optimization of variable structure modelica models using custom annotations. In Francesco Casella and Dirk Zimmer, editors, *Proceedings of the 7th International Workshop on Equation-Based Object-Oriented Mod-*

- eling Languages and Tools - EOOLT '16*, pages 45–54, New York, New York, USA, 2016. ACM Press. ISBN 9781450342025. doi:10.1145/2904081.2904088.
- Daniel Bender. Integration of exergy analysis into model-based design and evaluation of aircraft environmental control systems. *Energy*, 137:739–751, 2017. ISSN 0360-5442. doi:10.1016/j.energy.2017.05.182.
- Hilding Elmqvist, Hubertus Tummescheit, and Martin Otter. Object-oriented modeling of thermo-fluid systems. In The Modelica Association, editor, *Proceedings of the 3rd International Modelica Conference*, pages 269–286, 2003.
- F-Chart. Ees, 2018. URL <http://www.fchart.com/ees/>.
- Stefan Golle, Ullrich Hesse, Enrico Klausner, Frank Klimpel, Hans Brunswig, and Mario Raddatz. Betriebsphasenabhängig steuerbare flugzeugklimaanlage und verfahren zum betreiben einer derartigen flugzeugklimaanlage, 2016.
- P. Grassmann. Zur allgemeinen definition des wirkungsgrades. *Chemie Ingenieur Technik - CIT*, 22(4):77–80, 1950. doi:10.1002/cite.330220402.
- Matthias Hellerer and Fabian Buse. Compile-time dynamic and recursive data structures in modelica. In Dirk Zimmer and Bernhard Bachmann, editors, *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools - EOOLT '17*, pages 81–86, New York, New York, USA, 2017. ACM Press. ISBN 9781450363730. doi:10.1145/3158191.3158205.
- Andrea Lazzaretto and George Tsatsaronis. Speco: A systematic and general methodology for calculating efficiencies and costs in thermal systems. *Energy*, 31(8-9):1257–1289, 2006. ISSN 0360-5442. doi:10.1016/j.energy.2005.03.011.
- Dirk Limperich, Marco Braun, Kathrin Prölb, and Gerhard Schmitz. System simulation of automative refrigeration cycles. *Proceedings of the 4th International Modelica Conference*, 2005.
- Andreas Pfeiffer. Optimization library for interactive multi-criteria optimization tasks. In: *Proceedings of the 9th International Modelica Conference*, 2012.
- Roohbeh Sanghi, Jahangiri Pooyan, Alexander Thamm, Rita Streblow, and Dirk Müller. Dynamic exergy analysis - part i: Modelica-based tool development. *Building Simulation and Optimization Conference*, 2014.
- Michael Sielemann, Tim Giese, Bettina Oehler, and Martin Otter. A flexible toolkit for the design of environmental control system architectures. In: *Proceedings of the First CEAS European Air and Space Conference*, 2007. Berlin.
- The Modelica Association. The modelica language specification: Version 3.3, 2013. URL <https://www.modelica.org/documents/ModelicaSpec33Revision1.pdf>.
- G. Tsatsaronis and T. Morosuk. Exergy-based methods applied to the chain natural gas – lng – natural gas. *Proceedings of the 3rd International Exergy, Life Cycle Assessment, and Sustainability Workshop & Symposium (ELCAS3)*, July 2013.
- George Tsatsaronis. Definitions and nomenclature in exergy analysis and exergoeconomics. *Energy*, 32(4):249–253, 2007. ISSN 0360-5442. doi:10.1016/j.energy.2006.07.002.
- Michael Wetter, Wangda Zuo, Thierry S. Noudui, and Xiufeng Pang. Modelica buildings library. *Journal of Building Performance Simulation*, 7(4):253–270, 2014. ISSN 1940-1493. doi:10.1080/19401493.2013.765506.
- Stefan Wischhusen, Bruno Lüdemann, and Gerhard Schmitz. Economical analysis of complex heating and cooling systems with the simulation tool hksim. In: *Proceedings of the 3rd International Modelica Conference*, 2003.
- Dirk Zimmer and Daniel Schlabe. Implementation of a modelica library for energy management based on economic models. In The Modelica Association, editor, *Proceedings of the 9th International Modelica Conference*, pages 133–142, 2012. doi:10.3384/ecp12076133.
- Dirk Zimmer, Daniel Bender, and Alexander Pollok. Robust modeling of directed thermofluid flows in complex networks. In: *Proceedings of the 2nd Japanese Modelica Conferences*, 2018.

On Closure Relations for Dynamic Vapor Compression Cycle Models

Christopher R. Laughman Hongtao Qiao

Mitsubishi Electric Research Laboratories
Cambridge, MA, USA

{laughman, qiao}@merl.com

Abstract

Models of closure relations, or the expressions that relate the heat transfer coefficients and frictional pressure losses to other variables of the vapor-compression cycle, can have a significant impact on the performance on the overall cycle behavior. We explore three different approaches that may be used in formulating these closure models, and show that approaches that impose a nonlinear algebraic coupling can impose significant computational challenges. In comparison, models that incorporate low-pass dynamics can effectively decouple this nonlinear behavior, resulting in simulations that are faster and demonstrate more realistic and robust behavior.

Keywords: Modelica, heat pump, vapor compression cycle, approximation

1 Introduction

At the heart of modeling lies the art of approximation. Models of physical phenomena are always driven by a set of requirements that may relate to an exploration of possible system architectures, designing controls, or a variety of other possible purposes. These requirements drive the formulation of the model, so that the complexity and computational speed of the model must be balanced against the accuracy requirements.

This tradeoff can be readily observed from even a cursory survey of the two dominant methodologies for dynamically modeling heat exchangers: moving boundary models and finite volume models. These methodologies have different levels of complexity and requirements for simulation time: finite volume models describe the heat exchanger behavior accurately at a fine spatial and temporal resolution at a large cost of simulation time, while moving boundary models lump the spatial behavior of heat exchangers into a limited set of up to three fluid zones, and are correspondingly fast. Both of these methodologies are appropriate for different purposes, and the modeling engineer will usually choose the fastest approach with sufficient accuracy according to the specific requirements of the application.

The description of local heat transfer coefficients (HTCs) and frictional pressure losses (commonly referred to as closure relations, due to the fact that they "close" the

system of equations so that the number of equations equals the number of variables) in these models can be particularly challenging, as the correlations developed to most accurately describe experimentally observed phenomena are usually formulated with accuracy as the primary concern, and with little regard for computational considerations. Consequently, they can be difficult to incorporate into system-level models of thermofluid systems as they may be extremely nonlinear, tend towards infinity as mass flow rates go to zero, or exhibit other problematic behavior. These correlations are also usually defined only for specific flow conditions or refrigerant phases, so that there will inevitably be significant discontinuities between regions of the validity for specific correlations. Moreover, dynamic simulation presents additional difficulties as the unknown refrigerant mass flow rates, pressures, and specific enthalpies preclude the use of any initial information about the phase of the refrigerant (condensation, evaporation, liquid, or vapor) or the flow regime (laminar or turbulent), so the correlations must be defined in a manner which encompasses a wide range of flow conditions.

A variety of different approaches have previously been proposed to manage these closure relations both for steady-state and dynamic simulation. Perhaps the most straightforward of these approaches involves the creation of simplified correlations with improved behavior that have parameters that are tuned to approximate the original correlations. This method is used in a wide variety of literature (Qiao et al., 2015), and results in simulations that match observed experimental behavior quite well. Cycle models, in particular, require formulations of HTCs or frictional pressure losses that cover wide ranges of flow conditions, and the use of interpolation methods to smoothly stitch together correlations governing specific sets of conditions across transition regions has been used quite successfully (Elmqvist et al., 2003). An alternative approach was also proposed by Laughman et al. (2016) in which a nonparametric kernel regression method was used to approximate heat transfer coefficients directly from data, and the resulting simulations were shown to work well.

Despite the general success of these methods, simulations of vapor compression cycle models that employ these approaches can still exhibit problematic dynamics

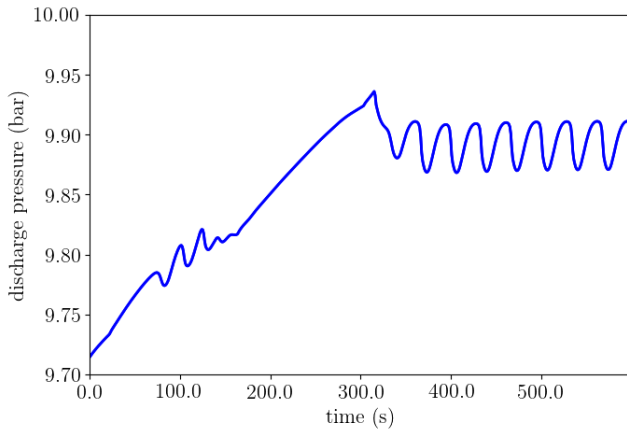


Figure 1. Oscillations in compressor outlet pressure after initialization with constant inputs.

that result in slow execution times and demonstrate non-physical behavior. One example is provided in Figure 1, which illustrates a periodic oscillation in the compressor outlet pressure that can arise when the system of differential algebraic equations (DAEs) representing the cycle behavior is initialized and driven with constant inputs. These oscillations, which can occur in either the presence or absence of any external forcing function, are related to interactions between a simplified HTC model and the rest of the system dynamics, and will be discussed further in Section 3. This spurious behavior, which is not observed in experiments, will affect many other variables in the cycle and significantly increase simulation times. Because these oscillations can be shown to be related to the closure models used, alternative approaches for describing the heat transfer coefficient and frictional pressure drop may be of interest to the modeling engineer.

In this work, we develop models of the vapor compression cycle in Modelica (Modelica Association, 2017) to investigate the effect that different closure relation models have on the overall system, and demonstrate that the addition of low-pass dynamics to the closure models can significantly improve the performance of the cycle simulations, particularly for the heat transfer coefficient. In Section 2, we develop models of the components used in the vapor compression cycle, and then study the effect of the original closure models on the overall cycle dynamics as well as the modified closure models with the added dynamics in Section 3. Section 4 provides a brief treatment of some results indicating the efficacy of these methods, and then a brief set of concluding remarks is presented in Section 5.

2 Component & System Models

We focus on the simulation of simple vapor compression cycles in this work, such as are used in many contemporary air-conditioning and heat pumping systems; a schematic illustrating a prototype cycle that includes a condensing tube-fin heat exchanger (HEX), an evaporat-

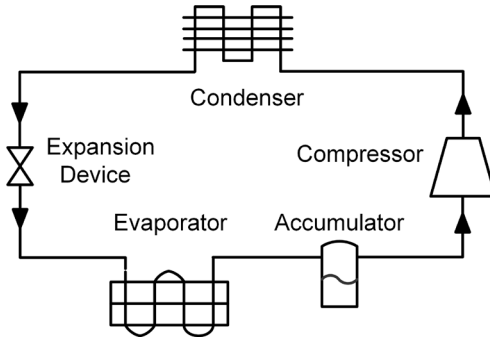


Figure 2. Basic vapor compression cycle.

ing tube-fin HEX, a compressor, and a linear expansion valve (LEV) is illustrated in Figure 2. Because the temporal behavior of the cycle is dominated by the HEXs over the time scales of interest, the system models in this work used dynamic models of the HEXs and static (algebraic) models of the compressor and expansion valves. Finite volume models (Li et al., 2014) were used for the HEXs to capture the dynamic behavior of the refrigerant pressures, as well as the spatially-dependent characteristics of these components. We assume 1-D refrigerant flow so that properties only vary along the length of the pipes; we also assume that the refrigerant can be described as a Newtonian fluid, negligible viscous dissipation and axial heat conduction in the direction of flow, negligible contributions to the energy equation from the kinetic and potential energy of the refrigerant, negligible dynamic pressure waves in the momentum equation, and thermodynamic equilibrium in each two-phase refrigerant volume.

Under these assumptions, the partial differential equations describing the conservation of mass, momentum, and energy (Levy, 1999) for the refrigerant can be spatially discretized for these finite volume models. A staggered grid scheme, illustrated in Figure 3, is used to avoid nonphysical pressure variations caused by numerical artifacts by decoupling the mass and energy equations computed for the volume cells (represented by the black solid boundary) from the momentum equations computed for the flow cells (represented by the red dashed boundary). Integration of these equations across these cells, as well as the use of the upwind difference method to approximate refrigerant properties for the convection-dominated flows from this application, results in a set of ordinary differential equations describing the conservation equations, as given in Equations 1, 2, and 3.

$$A_c \Delta z \rho_i = \dot{M}_{i-1/2} - \dot{M}_{i+1/2}, \quad (1)$$

$$\Delta z \frac{d\dot{M}_{i+1/2}}{dt} = \dot{I}_i - \dot{I}_{i+1} - A_c (p_{i+1} - p_i) - P \Delta z \bar{\tau}_{w,i+1/2}, \quad (2)$$

$$A_c \Delta z u_i = \dot{M}_{i-1/2} (h_{i-1/2} - \bar{h}_{\rho,i}) - \dot{M}_{i+1/2} (h_{i+1/2} - \bar{h}_{\rho,i}) + P \Delta z q_i'', \quad (3)$$

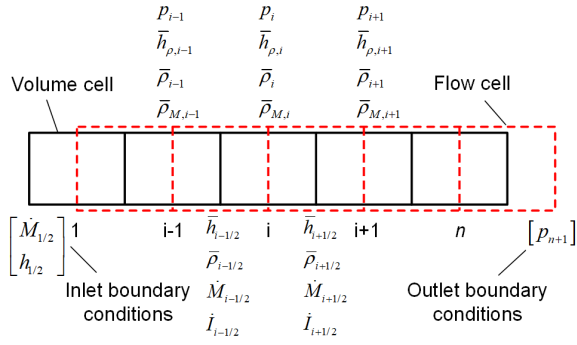


Figure 3. Finite volume discretization of refrigerant pipe.

where $\bar{\rho}_M$ represents the momentum density, \bar{h}_ρ and \bar{h} signify the density-weighted and flow-weighted specific enthalpies, the wall shear stress $\bar{\tau} = \frac{1}{2}f\bar{\rho}u|u|$ and f is the Fanning friction factor, and P is the circumference of the flow channel. The closure model for the frictional pressure term f will be provided in Section 3, and symbols with overbars represent average quantities in each cell. The dynamic states used in this model include the refrigerant pressures p , density-weighted specific enthalpies \bar{h}_ρ , and densities ρ (Laughman et al., 2017).

Because the fully dynamic momentum balance adds considerable complexity to the description of the system so that the system description consists of $3n$ ODEs where there are n volumes per HEX, a number of different approximations to the momentum balance have been developed to reduce this complexity and speed up these simulations (Qiao and Laughman, 2018). We used three variants in this study, including a friction-only formulation, the assumption of a uniform dp/dt , and the assumption of a linear pressure distribution. In the friction-only formulation, we assume that the derivative in Equation 2 is negligible due to the inertia term's minor importance to the thermal behavior of the system, and also neglect the acceleration pressure loss and gravity effect (Brasz and Koenig, 1983), since both of them are typically smaller than the frictional pressure loss. This yields a momentum balance of the form

$$p_{i+1} = p_i - \frac{P}{A_c} \Delta z \bar{\tau}_{w,i+1/2}. \quad (4)$$

In this formulation, we assume that the number of state variables are still $2n$ and the mass flow rates are the algebraic variables.

In the uniform dp/dt approach, the fact that the acoustic waves propagate with a speed of sound in the direction of fluid flow is used to motivate the assumption that the time derivatives of pressures are spatially invariant along the direction of flow. As a result, the number of dynamic pressure states is reduced to a single numerical state per pressure level, yielding a very efficient system of equations for each control volume, since dp/dt is given as an input. Note that the time derivative of pressure is not treated as constant over time, but rather in space along the

direction of flow. The pressure distribution in the heat exchanger still depends on the selected pressure loss models, and thus on the mass flow rates (Richter, 2008).

Since $\frac{dp_i}{dz} = \frac{dp_{n+1}}{dz} = \frac{dp_{ref}}{dz}$, the equations of continuity and energy become

$$A_c \Delta z \left(\frac{\partial \bar{\rho}_i}{\partial p_i} \frac{dp_{ref}}{dt} + \frac{\partial \bar{\rho}_i}{\partial \bar{h}_{\rho,i}} \frac{d\bar{h}_{\rho,i}}{dt} \right) = \dot{M}_{i-1/2} - \dot{M}_{i+1/2} \quad (5)$$

$$A_c \Delta z \left(\bar{\rho}_i \frac{d\bar{h}_{\rho,i}}{dt} - \frac{dp_{ref}}{dt} \right) = \dot{M}_{i-1/2} (h_{i-1/2} - \bar{h}_{\rho,i}) - \dot{M}_{i+1/2} (h_{i+1/2} - \bar{h}_{\rho,i}) + P \Delta z q_i'' \quad (6)$$

In this approach, only specific enthalpies are the differential variables, and therefore the total number of dynamic states is reduced from $3n$ by $n - 1$.

In comparison, the linear pressure loss approach assumes that the pressure is linearly distributed along the heat exchanger (Jensen, 2003), i.e., $p_i = p_1 + (i - 1) \frac{p_{n+1} - p_1}{n}$. Therefore, one can obtain

$$\frac{dp_i}{dt} = \left(1 - \frac{i-1}{n}\right) \frac{dp_1}{dt} + \frac{i-1}{n} \frac{dp_{n+1}}{dt}. \quad (7)$$

Aggregating all local momentum balances for flow cells 1 to n results in

$$n \Delta z \frac{d\bar{M}}{dt} = \dot{I}_1 - \dot{I}_{n+1} - A_c (p_{n+1} - p_1) - P \Delta z \sum_{i=1}^n \bar{\tau}_{w,i+1/2} - g A_c \Delta z \sum_{i=1}^n \bar{\rho}_{i+1/2} \sin \theta_i \quad (8)$$

where \bar{M} is the average mass flow rate $\frac{1}{n} \sum_{i=1}^n \dot{M}_{i+1/2}$. This variant can be further simplified by summing up the momentum equation with only frictional pressure loss instead for all the flow cells. This variant only has $n + 3$ dynamic states.

The refrigerant wall is modeled as one-dimensional heat conduction in the direction perpendicular to the refrigerant flow, with convective boundary conditions described by the refrigerant-side and air-side heat transfer coefficients, which will also be given in Section 3. This wall element can be modeled simply by

$$\frac{d(M_w c_w)}{dt} = \frac{k_w A_s (T_a - T_w)}{L_w/2} + \frac{k_w A_s (T_b - T_w)}{L_w/2}, \quad (9)$$

and the surface wall temperatures T_a and T_b are related to the bulk temperature of the adjacent fluid by

$$Q = \alpha A (T_{fluid} - T_{surf}). \quad (10)$$

A multicomponent ideal gas moist-air model was used for the air-side of this work. The mass and energy conservation equations used to describe the heat transfer from the outer surface of the tubes to the air reflected this multicomponent model, as described by Equation 12, where the

mass transfer coefficient was given by a modified Lewis correlation.

$$\dot{M}_{air} c_{p,air} \frac{dT_{air}}{dy} \Delta y = \alpha_{air} (A_{o,tube} + \eta_{fin} A_{o,fin}) (T_w - T_{air}) \quad (11)$$

$$\dot{M}_{air} \frac{d\omega_{air}}{dy} \Delta y = \alpha_m (A_{o,tube} + \eta_{fin} A_{o,fin}) \times \min(0, \omega_{water,sat} - \omega_{air}) \quad (12)$$

A simple isenthalpic model of the electronic expansion valve was also used, as described by a standard orifice flow equation

$$\dot{M} = C_v a_v \sqrt{\rho_{in} \Delta P}, \quad (13)$$

where the mass flow rate is regularized in the neighborhood of zero flow to prevent the derivative of the mass flow rate from tending toward infinity. The flow coefficient C_v is generally determined via calibration against experimental data, while the flow area a_v represents the control authority over the orifice size.

All cycle models in this work included a variable-speed high-side rotary compressor. We used simplified 1-D models of this component to describe the system due to the complex nature of the heat transfer and fluid flow through the compressor. Its performance was described by relating the volumetric efficiency η_v and isentropic efficiency η_{is} to the suction pressure P_{suc} , discharge pressure P_{dis} , and compressor frequency f , as given by

$$\eta_v = \frac{\dot{M}_{comp}}{\rho_{suc} V f} \quad (14)$$

$$\eta_{is} = \frac{h_{dis,isen} - h_{suc}}{h_{dis} - h_{suc}}. \quad (15)$$

The compressor power consumption \dot{W} was also related to the compressor speed and the ratio of inlet and outlet pressures, i.e., $\dot{W}(P_{rat}, \omega)$. The coefficients used for the functional forms of η_v , η_{is} , and \dot{W} were derived from experimental data, and the expressions themselves are provided in (Laughman et al., 2017).

Standard fan laws (ASHRAE, 2008) were used to describe the behavior of the heat exchanger fans, in which the volumetric flow rate was assumed to be directly proportional to the fan speed, while the power consumed by the fan was assumed to be proportional to the cube of the fan speed. These simple algebraic models were scaled by experimentally measured values of fan speed, flow rate, and power for a representative system; to minimize the error in these fits, linear and quadratic terms were also included in the power model to account for observed variations in the data.

A simple room model was also used to study the dynamics of the different cycle models on the idealized model of an occupied space. A lumped model of the room air was used to describe the sensible and latent dynamics of the space, and the room model was coupled to the

| Parameter | Value |
|------------------------------------|-------|
| Refrigerant | R134A |
| Total refrigerant mass (kg) | 1.65 |
| condensing HEX tube diameter (mm) | 7.9 |
| evaporating HEX tube diameter (mm) | 6.3 |
| condensing HEX tube length (m) | 0.5 |
| evaporating HEX tube length (m) | 2.5 |
| condensing HEX number of tubes | 27 |
| evaporating HEX number of tubes | 10 |

Table 1. Geometric parameters of the vapor compression cycle under consideration.

ambient environment through a simple RC circuit model of a building envelope with convective heat transfer on both the outside and inside interfaces of the envelope. Selected information about some of the important geometric parameters of the system is provided in Table 1.

3 Closure Models

In the previous section, we described the component-based models of the conservation equations that describe the behavior of all of the components: the heat exchangers, the expansion valve, and the compressor. As suggested in the introduction, however, the closure models used to relate the heat transfer coefficient and the frictional pressure loss to other variables in the cycle can play an important role in the overall cycle dynamics. We therefore study three types of closure models in this section: full algebraic models that are developed directly from correlations published in the literature, simplified algebraic models that are related to the previous correlation-based models but have a much simpler mathematical form, and dynamic models that also include time-dependent effects. These methods will each be described in turn, and then the results of implementing each of them in a complete vapor-compression cycle will be discussed in Section 4.

3.1 Full algebraic models

The most straightforward approach to describing closure models involves the direct implementation of the original HTC and frictional pressure loss correlations from the literature. Because these variables are dominated by microscopic phenomena, first principles-based models are generally either too analytically difficult to formulate or too computationally difficult to simulate, so there is a strong tradition of measuring these phenomena experimentally and fitting mathematical formulas to the resulting data. Because the parametric dependence of these relations changes depending on the nature of the flow and the refrigerant, these correlations are usually formulated for specific flow conditions, such as single-phase, two-phase, laminar, and/or turbulent conditions, and are only valid for that type of flow. We used a variety of these correlations to describe the flow, as described in Table 2.

| Correlation Type | Correlation |
|---------------------------------|----------------------------------|
| 1 ϕ liquid HTC | Dittus and Boelter (1930) |
| Condensing HTC | Dobson and Chato (1998) |
| Boiling HTC | Gungor and Winterton (1987) |
| 1 ϕ vapor HTC | Dittus and Boelter (1930) |
| 1 ϕ laminar ΔP_f | Blasius (Stephan, 2010) |
| 1 ϕ turbulent ΔP_f | Hagen-Poiseuille (Stephan, 2010) |
| Condensing ΔP_f | Lockhart and Martinelli (1949) |
| Boiling ΔP_f | Jung and Radermacher (1989) |

Table 2. Correlations used to describe heat transfer coefficients and frictional pressure losses.

The nature of dynamic simulation makes it very difficult to ensure that a specific set of conditions will always be met, due to the challenges of initializing large systems of DAEs and the fact that the flow conditions may change dynamically so that a given cell in the finite volume HEX model sometimes experiences single-phase conditions and sometimes experiences two-phase conditions. In addition, it is computationally beneficial if the correlations are at least C^1 continuous over a wide range of operation, to allow the integrators to function efficiently. Discontinuities in the closure models may also require the integrator to do event iteration or generate nonlinear sets of equations to solve.

We therefore blend these correlations together via a univariate trigonometric interpolation method (Richter, 2008). In this method, the user defines transition zones between regions of validity for the individual correlations according to some given variable that will be used to determine which correlation to use. The output of the method is a C^2 continuous function that smoothly transitions between different correlations, depending on the value of the transition variable. For example, the condensing and boiling heat transfer coefficients are combined into a unified two-phase HTC that uses the value of the temperature difference between the refrigerant and the pipe wall to smoothly transition between these two flow regimes. If ΔT is greater than T_c degrees C, then the condensing heat transfer coefficient is used, while if ΔT is less than T_b degrees C, the boiling heat transfer coefficient is used. These values are smoothly interpolated in a transition region from $T_b < 0 < T_c$.

This unified two-phase HTC is then itself blended with the liquid and vapor heat transfer coefficients by using the flow quality x as the transition variable and transition regions on either side of the two-phase region. If $x < 0$ the liquid heat transfer coefficient is used, while if $0 < x < x_1$ then there is a smooth transition from the liquid heat transfer coefficient to the two-phase heat transfer coefficient. Similarly, if $x_2 < x < 1$ there is a smooth transition between the two-phase heat transfer coefficient and the vapor heat transfer coefficient, and the vapor heat transfer coefficient is used if $x > 1$. The resulting blended heat transfer coefficient then is able to accurately describe the

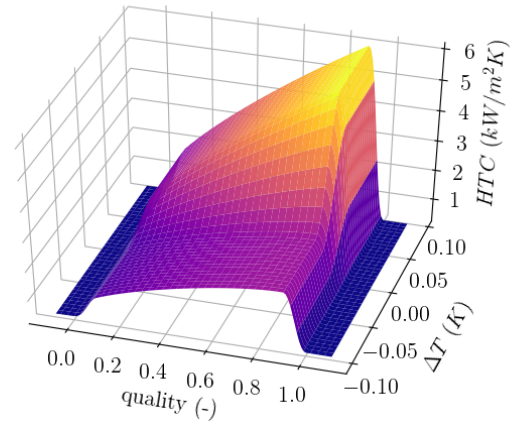


Figure 4. Blended HTC as a function of the wall temperature difference ΔT and the flow quality x .

flow over all flow regimes. A similar approach is also used to unify the different frictional pressure loss correlations into a blended correlation covering a wide range of flow regimes.

The efficacy of this approach can be seen in Figure 4, which illustrates the HTC as a function of the temperature difference ΔT between the refrigerant and the tube wall as well as the flow quality x of the refrigerant. This surface was mapped out by using the correlations provided in Table 2; the temperature transition region between boiling and condensation was set to $-0.025 \text{ K} \leq \Delta T \leq 0.025 \text{ K}$, and the transition regions between both single-phase regions and the two-phase region were set to 0.05 kg/kg . These transitions can be clearly seen by considering slices parallel to the quality axis: the HTC increases at the transition from the subcooled liquid to the two-phase regions, gradually continues to increase as the flow quality increases, and then rapidly decreases at the transition between the two-phase and superheated vapor region. Note the large slope of the transition between the two-phase and vapor regions, which will be discussed in more detail in the following sections.

While these somewhat Frankenstein-like equations might be said to most accurately represent the relation $\alpha_i = f(\cdot)$ according to the literature, the algebraic relations between the system inputs and the heat transfer coefficient α_i can be extremely nonlinear. When these functions are then compiled into a system model, these sets of equations are often solved using a nonlinear solver such as the multivariate Newton-Raphson method, which can be both time-consuming and fragile. Moreover, each volume in the heat exchanger will incorporate these nonlinear blocks, adding further complexity to the overall description of the system.

3.2 Simplified algebraic models

One alternative approach that has been successfully used to mitigate these nonlinearities has been the creation of simplified algebraic closure models that capture the gen-

eral trends of the detailed heat transfer coefficient and frictional pressure loss relations without implementing their complexity. A wide variety of forms can be used for these relations, depending on the required parametric dependence or level of fidelity to the behavior of the original correlations. For example, we used a simplified heat transfer relation for each phase according to

$$\alpha = \alpha_0 \left(\frac{\dot{M}}{\dot{M}_0} \right)^b. \quad (16)$$

The constants α_0 for the liquid, two-phase, and vapor flow regions were calculated by coarsely approximating the behavior of the full correlations over their regions of validity, and the same trigonometric interpolation method was used to smoothly transition between phases.

This method is powerful because it provides a small number of parameters to tune to the original correlations, as well as the widths of the transition regions, but the number of state variables that are coupled in these models is much smaller than is the case with the original correlations. This makes it possible to significantly improve the performance of the simulations by eliminating the large number of blocks of nonlinear equations. It is important to note that these simplified correlations still manifest very large changes in the heat transfer coefficient, due to the fact that the two-phase heat transfer coefficients can be more than an order of magnitude greater than the single-phase heat transfer coefficients. These large changes result in very large magnitudes in $d\alpha/dx$, which can also affect the system simulations.

Similar simplified formulations were also constructed for the frictional pressure loss, which was expressed as

$$\Delta p_f = K \Delta p_0 \left(\frac{\dot{M}}{\dot{M}_0} \right)^b, \quad (17)$$

where $b = 2$ for these models. The Colebrook correlation for the single-phase friction factor and the Friedel correlation for two-phase multipliers were used to determine the nominal values of K , Δp_0 , and \dot{M}_0 . This relation is not only less nonlinear than the original correlation-based relations, but it is also easily invertible and can allow the pressure loss to be calculated as a function of the mass flow rate, or vice versa. As such, the resulting systems simulations had much faster performance, since the nonlinear dependence on the variety of input variables was removed from the relation and the integrator could take much larger steps.

3.3 Dynamic models

The main challenges posed by the algebraic approaches used to formulate closure models in the previous section are the nonlinear sets of algebraic equations, which may couple together multiple state variables, and the potential effect of large gradients in the transition regions. The sets of nonlinear equations can present a particular problem

for the solvers, as the successful simulation of a model depends on the quality of nonlinear solvers used by a particular Modelica tool, as well as the initial guesses used by that method. This tool-dependent aspect of these models can potentially cause simulations of the same model in different tools to yield different answers, which is rather problematic.

Rather than implement these nonlinear algebraic relations, we propose the incorporation of dynamics into the closure models to decouple the HTC or frictional pressure loss from the other state variables. This will make the closure variables into state variables of the system, and will decouple the value of the closure variable in the fluid computations with the value of the closure variable calculated from the other state variables. In the case of the heat transfer coefficient, this may be calculated by

$$\hat{\alpha} = f(\dot{M}, p, h) \quad (18)$$

$$\frac{d\alpha}{dt} = \frac{1}{\tau} (\hat{\alpha} - \alpha), \quad (19)$$

where f denotes the algebraic HTC correlation, $\hat{\alpha}$ represents the algebraic HTC, and α represents the low-passed version of the HTC. Since this heat transfer coefficient is now a state variable, it needs to be initialized to an initial value α_0 at $t = 0$. We expect that this will eliminate many of the nonlinear equation blocks and also reduce the sensitivity of the system to the large gradients, as the changes in the heat transfer coefficient will now also depend on its previous values. The parameter τ should be tuned to be substantially faster than other time constants of the system in order to ensure that it will not change the system response.

These low-pass dynamics are often reasonable to include because the bandwidth of experimental vapor compression cycles is often much lower than is observed in simulations. Many potential sources of modeling error could potentially contribute to this discrepancy. For example, the working fluid in many practical cycles is a mixture of both refrigerant and oil, which is needed for lubrication of the compressor. The presence of this oil in heat exchangers will damp sudden changes in the refrigerant state, as refrigerant must diffuse into or out of the oil in response to changes in the refrigerant properties. In addition, these systems often have many time delays and couplings, such as axial heat conduction and diffusion, that are often not incorporated into dynamic models but are inherently low-pass in nature. In general, we do not advocate blindly adding dynamics to existing system models because these added dynamics will be convolved with original dynamics, but the tradeoff between inaccuracies of the added dynamics and the inaccuracies of simplified methods, not to mention the improvement in computational speed, may motivate the use of these approximations in some cases.

The implementation in Modelica is exceedingly simple, and is attractive from an object-oriented design perspec-

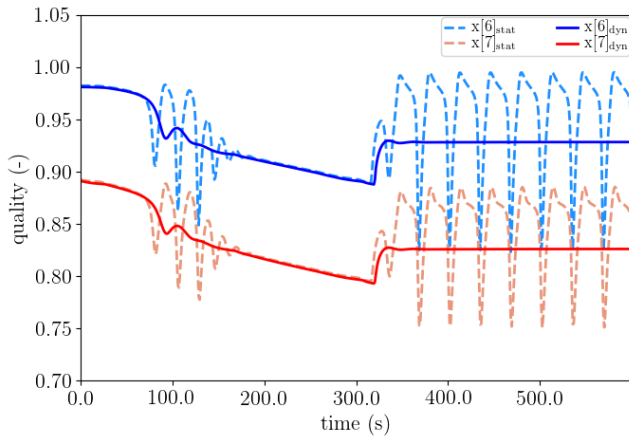


Figure 5. Refrigerant flow quality x in volumes 6 and 7 for simple algebraic HTCs and detailed dynamic HTCs.

tive. The code in the original correlation function that assigns the closure variable,

```

alphas[k] = simpleHTC_FullRegion(xs[k],
    alpha_vap, alpha_2ph, alpha_liq,
    xdot_1, xdot_2);

```

need only be changed to incorporate these dynamics in the output, e.g.,

```

alphaHat[k] = simpleHTC_FullRegion(xs[k],
    alpha_vap, alpha_2ph, alpha_liq,
    xdot_1, xdot_2);
der(alphas[k]) = (alphaHat[k]-alphas[k])
    /tau;

```

and the additional state equations will be incorporated into the model. This type of modification would be much more difficult to implement in a non equation-oriented language, as it would necessitate the complete rearrangement of the state vector.

4 Results

The effect of these different closure relations was studied by implementing the full vapor-compression cycle model in the Modelica language and comparing the resulting cycle behavior when different closure models were incorporated into the complete cycle model. These simulations were performed on a desktop with an Intel i7 processor with 32 Gb of RAM using the Dymola 2018 FD01 compiler (Dassault Systemes, AB, 2018), and the differential algebraic equations were integrated with the DASSL solver with the tolerance set to 10^{-5} . All of the models for this application were developed by the authors with the exception of the refrigerant property models, which were obtained from the commercial Vapor Cycle Library (Modelon AB, 2018).

As the cycle behavior with the alternate heat transfer coefficient models differs from the behavior with the alternate frictional pressure drop models, we discuss each of

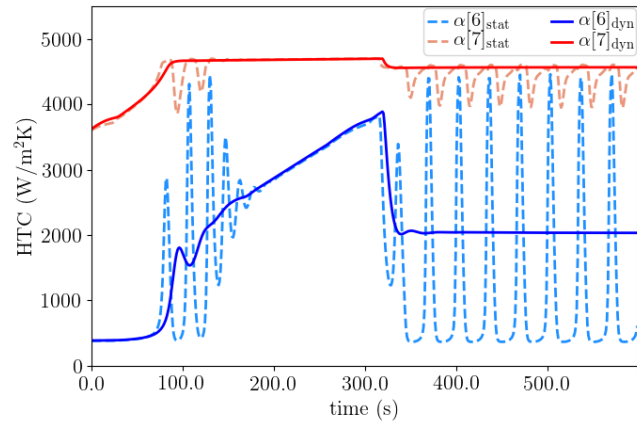


Figure 6. Refrigerant-side heat transfer coefficients α_i for volumes 6 and 7 for simple algebraic HTCs and detailed dynamic HTCs.

these types of closure models in their own respective subsections. While the base cycle model was identical in all of these experiments, the friction-only momentum balance was implemented with the simplified algebraic pressure drop model when heat transfer coefficient models were studied, while the simplified dynamic heat transfer coefficient model was used when the different pressure drop models were studied.

4.1 Heat transfer coefficient models

We first investigated the use of the heat transfer coefficient model based upon the algebraic correlations from the literature, as described in the beginning of Section 3.1. While the Modelica tool was able to compile these models, it was unable to initialize or run simulations of these models due to failure of convergence for the nonlinear equation solver. While detailed information about the flattened model from `dsmodel.mof` is unavailable due to the use of encrypted refrigerant property models, statistics from the compilation process provide some indication of the cause of this failure: this information indicates that 37 numerical Jacobians are generated, which corresponds to the number of refrigerant volumes included in the cycle (27 volumes for the condensing heat exchanger and 10 volumes for the evaporating heat exchanger, corresponding to the number of tubes in each). As suggested in Section 3, the algebraic equations that couple the heat transfer coefficient to the pressure, specific enthalpy, and mass flow rate in each volume are quite nonlinear and difficult to solve. While it may be possible to customize this correlation or otherwise provide sufficient information to the compiler (e.g., analytical derivatives via annotations or external functions) for the nonlinear solver to function correctly, the extra effort to do so for a specific set of correlations could be substantial, and would have to be replicated for every new correlation.

As the cycle models incorporating the correlation-based HTC models did not produce any viable simulations, we

turned our attention to the use of the simplified HTC models. While the reduced nonlinearity of these closure models enabled the successful simulation of the cycle behavior, oscillations in the cycle behavior as seen in Figure 1 can occur due to either an input forcing function or simply the selection of an unlucky operating point. These oscillations are caused by the high gain of $d\alpha/dx$ in the transition region; small changes in the state variables are coupled to large changes in the heat transfer coefficient, which in turn has a measurable effect on the mass flow rate, pressures, and specific enthalpies. In this situation, these interactions can act to drive the system towards an unstable steady-state manifold and result in limit cycle behavior.

This behavior can be seen in Figures 5 and 6, which illustrate the flow quality and local refrigerant-side heat transfer coefficient in volumes 6 and 7 in the condenser for both the simplified algebraic HTC and the simplified dynamic HTC. For this heat exchanger, the HTC model is configured to have a liquid heat transfer coefficient of $378 \text{ W/m}^2\text{K}$ below a flow quality of 0, a two-phase heat transfer coefficient of $4600 \text{ W/m}^2\text{K}$ between a flow quality of 0.1 and 0.9, and a vapor heat transfer coefficient of $369 \text{ W/m}^2\text{K}$ above a flow quality of 1, all at a mass flow rate of 36 g/s . It is evident that the algebraic HTCs in Figure 6 change abruptly when the flow quality illustrated in Figure 5 for either volume moves into the transition region. More specifically, the large spikes in the heat transfer coefficient for volume 6 are directly associated with the excursions in the transition region. These large gradients in the heat transfer coefficient are directly coupled to many other aspects of the flow. As the heat transfer coefficients change rapidly, the flow itself changes in such a way to reduce the flow quality and form this periodic behavior.

These oscillations in the heat transfer coefficient and flow quality can affect the overall cycle simulation in a variety of ways. Aside from the corresponding oscillations in other variables that are coupled to the heat transfer coefficient, these large gradients will also prevent the integration routines from taking large time steps, resulting in long simulation times. These effects can be mitigated somewhat for these simplified algebraic closure models in many circumstances by increasing the width of the transition regions, but the resulting gradual changes in the heat transfer coefficients can pose problems when linearizing models for control design (due to the high gains from the derivatives of the heat transfer coefficients) and also may not be justified by experimental data.

In this context of challenges posed by these algebraic heat transfer coefficients, we evaluated the performance of the vapor compression cycle with the dynamic heat transfer coefficient models. This study was performed by directly replacing the heat transfer model in the previous set of cycle models using the `redeclare` keyword, thus enabling a direct comparison of the performance of the method. The time constant τ of the dynamic heat transfer coefficient models was set to 3 seconds, though other values were also used successfully; in general, this time

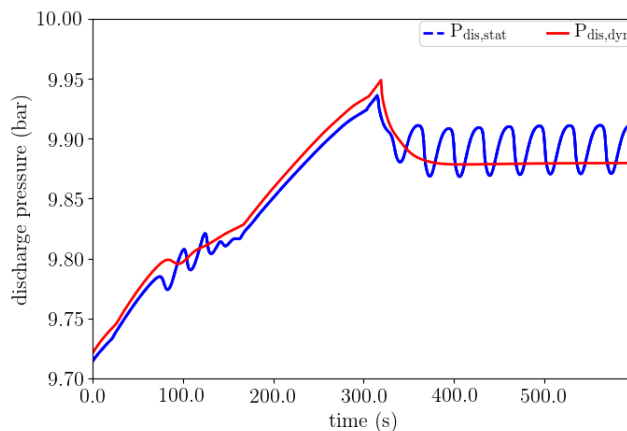


Figure 7. Compressor discharge pressure using simple heat transfer coefficient model with and without low-pass dynamics.

constant should be set to a relatively small value to minimize its affect on other cycle dynamics.

These new cycle simulations did not exhibit any oscillations, as can be seen in Figures 5 and 6, and also did not have any of the numerical Jacobians seen in the first set of algebraic heat transfer coefficient models. The low-pass behavior of the heat transfer coefficient has the effect of damping out these oscillations, allowing the system to converge to a steady-state operating point. Comparing both the flow quality and the local heat transfer coefficients for both volumes 6 and 7, the filtered value of these variables can be seen to be within the range of values for the previous simulations, suggesting that the cycle behavior with the dynamic closure model is not affected significantly. Moreover, Figure 7 illustrates behavior of the compressor discharge pressure both with and without the dynamic HTC model, confirming both that the oscillations in this pressure signal were caused by the heat transfer coefficient and that the dynamic HTC can successfully eliminate these oscillations. The addition of these dynamics had a minimal effect on the overall cycle simulations; the RMSE between the cycles with and without the dynamic HTCs was only 2.7 W out of a total cooling capacity of 1190 W .

Table 3 illustrates the significant benefits that these dynamic HTC models can have on the cycle simulation time. While the correlation-based algebraic models were unable to simulate, a 1000 second simulation of the cycle with simplified algebraic models took 228 seconds to run due to the oscillations caused by the heat transfer coefficient model. Use of the dynamic HTC models reduced this computational time significantly; the dynamic version of the correlation model could be used and had a CPU time of 146 seconds, while the dynamic version of the simplified HTC model was able to simulate the entire 1000 second duration in 39 seconds, representing an 83% reduction in CPU time. These results suggest that multiple HTC models could be used profitably in simulating the cycle; the simplified models could be used first to quickly study large-scale dynamics, and the more detailed correlations

| HTC Formulation | CPU Time (s) |
|------------------------------|--------------|
| Algebraic, correlation-based | – |
| Algebraic, simplified | 228 |
| Dynamic, correlation-based | 146 |
| Dynamic, simplified | 39 |

Table 3. CPU time for alternate HTC formulations.

could then be used to obtain more accurate results.

4.2 Pressure loss models

The models of the pressure drop were tested in an analogous manner to those of the heat transfer coefficient. The base vapor-compression cycle model was used with the simplified dynamic heat transfer coefficient model, and the performance of the system with the friction-only, dp/dt , and linear pressure drop momentum balances was evaluated with both algebraic and dynamic closure models. Although the dp/dt and linear pressure drop models neglect the frictional pressure loss term, the method described in Section 3 was used to decouple the mass flow rate from the pressure drop in each volume.

In general, frictional pressure drop correlations are written as $dp = f(\dot{M})$, so that the mass flow rate must be known to calculate the effective pressure drop. In correlations where dp is a function of both \dot{M} and other thermodynamic variables (e.g., $dp = f(Re, \sigma, \dots)$), these correlations will result in a set of nonlinear equations, and the addition of dynamics into the frictional pressure drop model will improve their performance in much the same way that the heat transfer coefficient models were improved. In these cases, we have found that the value of the time constant τ must be relatively small to avoid influencing the system dynamics because the mass flow rate exhibits relatively high frequency behavior with respect to the fastest dynamics of the cycle. Unfortunately, the dynamics of the modified system approach those of the unmodified system as τ is reduced, so that the simulation speed of a cycle model with very fast dynamic closure models will be nearly identical to the simulation speed of a cycle model with algebraic closure models.

Unlike the heat transfer coefficient models, there are many frictional pressure drop correlations available in which this relationship can be analytically inverted, so that $\dot{M} = f(dp)$. In these correlations, the mass flow rate can be directly calculated from the frictional pressure drop, and there is no need to solve a nonlinear set of equations. This is the case for many of the correlations commonly used to describe frictional pressure drop, and is also the case for the correlations used in this paper. As a result, the addition of dynamics to the frictional pressure drop models does not improve the performance for many standard formulations of the momentum balance, including the friction-only momentum balance used in this work.

In contrast to the friction-only momentum balance, the dp/dt and linear pressure drop models described in Sec-

| Model type | CPU time (s) |
|-----------------|--------------|
| Standard | 2855 |
| Friction only | 111 |
| dp/dt | 214 |
| Linear pressure | 175 |

Table 4. CPU time for variants of momentum balance with dynamic frictional pressure drops.

tion 2 have a tight algebraic coupling between the pressure drop between consecutive volumes and the mass flow rate through these volumes. A set of nonlinear algebraic equations that couple the pressures, specific enthalpies, and mass flow rates thus result, with the rather predictable consequence (at this point) that the system becomes very difficult to solve. Experiments with these momentum balances resulted in the same behavior observed with the correlation-based algebraic heat transfer coefficient models, in which there were 37 numerical Jacobians and the Modelica tool was unable to simulate the model. With the addition of dynamics to the pressure drop terms, the cycle behavior could be simulated in the same conditions used to evaluate the heat transfer coefficient models; Table 4 shows that the dp/dt and linear pressure drop approximations of the momentum balance are significantly faster than the transient momentum balance, though they are roughly comparable to the friction-only momentum balance.

5 Concluding Remarks

In this work, we explored some of the computational considerations that relate to the implementation of heat transfer coefficient and frictional pressure drop models that are included in vapor-compression cycle simulations, and described three different modeling approaches that can potentially improve the speed of the overall cycle simulations at a low cost to physical accuracy. On the basis of our experience, we would recommend that heat transfer coefficient models include some simple dynamics, as this enables the direct use of correlation-based models, and the reduction in computational time is significant. In comparison, these methods do not improve the frictional pressure drop models dramatically except when simplified momentum balances such as the dp/dt or linear pressure drop methods are used.

There are a variety of directions in which this work could continue. These methods could conceivably be applied to off-cycle simulation, as the small mass flow rates and pressure variations tend to cause high-frequency oscillations in these cases that are difficult to simulate. In addition, it would also be valuable to consider the experimental characterization of the dynamics of heat transfer, rather than solely characterizing the steady-state behavior. This could potentially be correlated with other relevant phenomena for practical vapor-compression cycles, such as the effect of the oil circulating with the refrigerant.

References

- ASHRAE. *HVAC Systems and Equipment Handbook*. ASHRAE, Atlanta, GA, 2008.
- J.J. Brasz and K. Koenig. Numerical methods for the transient behavior of two-phase flow heat transfer in evaporators and condensers. *Numerical Properties and Methodologies in Heat Transfer*, pages 461–476, 1983.
- Dassault Systemes, AB. Dymola 2018 FD01, 2018.
- F.W. Dittus and L.M.K. Boelter. Heat transfer in automobile radiators of the tubular type. *University of California Publications in Engineering*, 2(13):443–461, 1930.
- M.K. Dobson and J.C. Chato. Condensation in smooth horizontal tubes. *Journal of Heat Transfer*, 120:193–213, Feb 1998.
- H. Elmqvist, H. Tummescheit, and M. Otter. Object-oriented modeling of thermo-fluid systems. In *3rd International Mod- elica Conference*. Linkoping, Sweden, 2003.
- K.E. Gungor and R.H.S. Winterton. Simplified general correlation for saturated flow boiling and comparisons of correlations with data. *Chem. Eng. Res. Des.*, 65:148–156, 1987.
- J.M. Jensen. *Dynamic modeling of thermo-fluid systems with focus on evaporators for refrigeration*. PhD thesis, Technical University of Denmark, Department of Mechanical Engineering, 2003.
- D.S. Jung and R. Radermacher. Prediction of pressure drop during horizontal annular flow boiling of pure and mixed refrigerants. *International Journal of Heat and Mass Transfer*, 32(12):2435–2446, 1989.
- C. Laughman, H. Qiao, and D. Nikovski. Kernel regression for the approximation of heat transfer coefficients. In *Gustav Lorentzen Natural Working Fluids Conference*, 2016.
- C. Laughman, H. Qiao, S.A. Bortoff, and D.J. Burns. Simulation and optimization of integrated air-conditioning and ventilation systems. In *Proceedings of the 15th IBPSA Conference*, pages 1824–1833, 2017.
- S. Levy. *Two-phase flow in complex systems*. New York: John Wiley & Sons, 1999.
- P. Li, H. Qiao, Y. Li, J.E. Seem, J. Winkler, and X. Li. Recent advances in dynamic modeling of HVAC equipment. Part 1: Equipment modeling. *HVAC&R Research*, 20(1):136–149, 2014.
- R.W. Lockhart and R.C. Martinelli. Proposed correlation of data for isothermal two-phase, two-component flow in pipes. *Chemical Engineering Progress Symposium Series*, 45:39–48, 1949.
- Modelica Association. Modelica specification, Version 3.4, 2017. URL www.modelica.org.
- Modelon AB. *Vapor Cycle Library User Guide*, 2018. v2.1.
- H. Qiao and C. Laughman. Comparison of approximate momentum equations in dynamic models of vapor compression systems. In *Proceedings of the 16th International Heat Transfer Conference*, 2018.
- H. Qiao, V. Aute, and R. Radermacher. Transient modeling of a flash tank vapor injection heat pump system - part I: Model development. *Int. J. Refrigeration*, 49:169–182, 2015.
- C.C. Richter. *Proposal of new object-oriented equation-based model libraries for thermodynamic systems*. PhD thesis, Technische Universität Braunschweig, Institut für Thermodynamik, 2008.
- Peter Stephan, editor. *VDI Heat Atlas*. Springer-Verlag, 2010.

Fast Calculation of Refrigerant Properties in Vapor Compression Cycles Using Spline-Based Table Look-Up Method (SBTL)

Lixiang Li¹ Jesse Gohl¹ John Batteh¹ Christopher Greiner² Kai Wang²

¹Modelon Inc, USA, {lixiang.li, jesse.gohl, john.batteh}@modelon.com

²Ford Motor Company, USA, {cgreiner, kwang37}@ford.com

Abstract

Refrigerant property calculation has a significant impact on the computational performance of vapor compression cycle simulations. This paper summarizes a Modelica implementation of Spline-Based Table Look-Up Method (SBTL) for fast calculation of refrigerant properties. External C functions are used for faster spline evaluation and inversion. Significant improvement in computation speed was observed without sacrificing accuracy. An SBTL property model of R134a is first validated against a highly accurate Helmholtz energy equation of state (EOS) model. Then the new model was tested rigorously from single function calls, to heat exchanger test bench, to system models of the vapor compression cycle in Modelon's Air Conditioning Library. Finally, an SBTL property model of R1234yf was used in a drive cycle simulation and a shutdown-startup test of two complex air conditioning system models developed at the Ford Motor Company. These system models are running more than twice the speed of the ones using Helmholtz energy EOS.

Keywords: Refrigerant Properties, Equation of State (EOS), Thermodynamic Modeling, Vapor Compression Cycle, Air Conditioning, Spline Interpolation, Computational Performance

1 Introduction

Dynamic simulations of vapor compression cycles often involve significant numbers of function calls to calculate properties of the working fluid. These calculations are typically performed using reference Helmholtz energy (multi-parameter) equation of state (EOS) (Tillner-Roth et al, 1994; Richter et al, 2011) to achieve high accuracy. Short formulation (Span et al, 2003) of Helmholtz energy EOS improves the computational performance, but it does not cover all popular refrigerants, e.g. R1234yf. In Modelon's Air Conditioning Library, both the reference Helmholtz EOS and short Helmholtz EOS are implemented for a wide range of refrigerants.

The two approaches mentioned above have a large impact on the vapor compression cycle simulation

speed. First, they have a complicated multi-parameter functional form, which is very costly to evaluate. Moreover, Helmholtz energy EOS uses density and temperature to determine the thermodynamic state, but the system models are usually described by pressure and enthalpy as dynamic states. As a result, internal iteration is needed when the property calculations are performed in a vapor compression cycle simulation.

To address these performance issues, different interpolation methods have been used to approximate refrigerant properties. Extensive literature reviews can be found in the reference (Laughman et al, 2012; Schulze, 2013; Aute et al, 2014) and thus not repeated in this paper. The Spline-Based Table Look-Up Method (Kunick et al, 2015) is chosen to approximate different refrigerant properties in this work because it possesses the following unique features:

- Equidistant grid
- Continuous first derivatives
- Analytic inverse
- Consistent phase boundary definition

The first three features are guaranteed by a specific type of quadratic/biquadratic spline (Späth, 1995). Equidistant grid eliminates the need for searching when evaluating the spline. C^1 continuity is a necessity since some thermodynamic properties are expressed as derivatives, e.g. specific heat capacity, isobaric expansion coefficient, etc. Analytic inverse provides consistent forward and backward calculations without numerical iterations. The last feature avoids chattering around the phase boundary during dynamic simulations. To summarize, the SBTL method takes both function evaluations and system modeling requirements into account, which makes it stand out among different spline interpolation methods for refrigerant property calculation.

Three features in our implementation are tailored for modeling of vapor compression cycle in Modelica: (1) One overall fit over the whole domain is used for 2D splines, instead of fitting several sub-domains. This is to balance data size (hence, loading time) and accuracy. (2) Minimum use of grid transformation. While transforming the grid can improve accuracy, it adds complexity to the implementation and increase the computational cost when taking derivatives and

inverting functions. (3) The use of external C functions for the spline evaluation, inversion and derivatives. These calculations are repeated many times in different property functions. Implementing them in C further accelerates the simulations.

The methodology and implementation of the SBTL are further discussed in Section 2. The formulation of the spline interpolation, the data generation process, and the Modelica model structure are covered. The test results of the SBTL property models of R134a and R1234yf are summarized in Section 3. We first compare function calls of the STBL model of R134a to short Helmholtz R134a model in the Air Conditioning Library. The comparisons are then carried out on a heat exchanger testbench and system models. Finally, we tested the SBTL model of R1234yf in a vapor compression cycle model in the Air Conditioning Library before we used it in complex AC system models from Ford described in Section 3.5. The results and performance are compared against the reference Helmholtz R1234yf model (for which short formulation is not available).

2 Methodology and Implementation of Spline-Based Table Look-Up Method (SBTL)

The key concepts of the SBTL method are illustrated in this section using an example of 1D spline. A complete description of the method can be found in the reference (Kunick et al, 2015). The data generation process is also covered here followed by the explanation of the property model structure.

2.1 Overall scheme of the spline functions

The SBTL method uses piece-wise quadratic/bi-quadratic splines to approximate the refrigerant properties. Equidistant grid is used for the spline. So, when evaluating the spline, the interpolating cell is easily known without searching in the whole domain. To enhance the accuracy of the interpolation, transformation can be used on both the independent and dependent variables. In this case, chain rule must be applied properly when calculating derivatives for the transformed variables.

2.1.1 Example of 1D spline and its inverse

The SBTL method distinguishes “node” from “knot”. A node is where we have raw data and where the spline intersects with the raw data points. A knot is where two adjacent pieces of splines meet, i.e. the first derivative of the splines are equal. Moreover, a node is the midpoint of two neighboring knots, as shown in Figure 1. For an equidistant series of nodes (or knots) and a given point of evaluation \bar{x} , the interval number where it is located is given by

$$i = \text{floor} \left(\frac{\bar{x} - \bar{x}_1^K}{\Delta \bar{x}} \right) \quad (1)$$

where \bar{x}_1^K is the first knot of the whole spline and $\Delta \bar{x}$ is the distant between neighboring nodes (or knots). The spline function can then be expressed as:

$$\bar{z}_{\{i\}}(\bar{x}) = \sum_{k=1}^3 a_{ik} (\bar{x} - \bar{x}_i)^{k-1} \quad (2)$$

where \bar{x}_i is the node in the i^{th} interval and a_{ik} are the spline coefficients in the i^{th} interval. The bar over the independent variable x and dependent variable z indicate that they are transformed variables, which will be discussed in more details in Section 2.1.2. Note that the grid is equidistant in terms of the transformed independent variable \bar{x} .

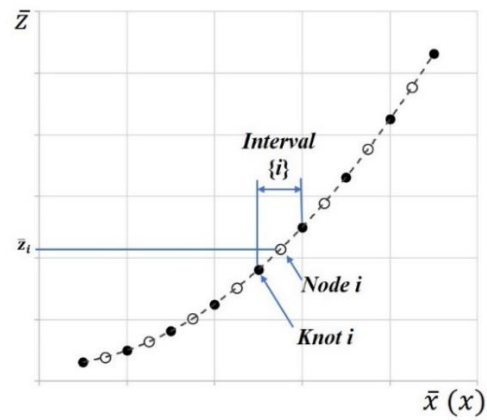


Figure 1. Illustration of the spline interpolation. Knots are represented by solid dots and node by hollow ones.

For a monotonic spline polynomial $\bar{z}_{\{i\}}(\bar{x})$ in the i^{th} interval, the inverse function of the spline is given as:

$$\begin{aligned} \bar{x}_{\{i\}}^{INV}(\bar{z}) \\ = \bar{x}_i + \frac{-a_{i2} \pm \sqrt{a_{i2}^2 - 4a_{i3}(a_{i1} - \bar{z})}}{2a_{i3}} \end{aligned} \quad (3)$$

where the sign (\pm) equals to $\text{sign}(a_{i2})$. An auxiliary spline function $\bar{x}_{\{i\}}^{AUX}(\bar{z})$ is also needed to estimate \bar{x} , so that we can locate the interval number i .

Spline interpolation in 2D is just an extension of the 1D example. The detailed formulation can be found in the reference (Kunick et al, 2015). The solution algorithm for the spline coefficients are given in the book (Späth, 1995).

2.1.2 Transformations and Derivatives

The equidistant grid enables us to calculate the interval number according to Equation 1, which removes the computational overhead of searching through the whole domain. However, when the function is highly non-linear, as shown in the left plot of Figure 2, we need to increase the number of nodes substantially to better approximate the function by a quadratic spline. This would end up in larger data files, especially for

2D splines (data file size $\sim O(n_{nodes}^2)$), and it would take more time to load the spline coefficient data at initialization.

To balance accuracy and data file size, proper transformations can be applied on both independent and dependent variables. For example, we transformed the pressure coordinate with (base 10) logarithmic function to get better resolution at low pressures.

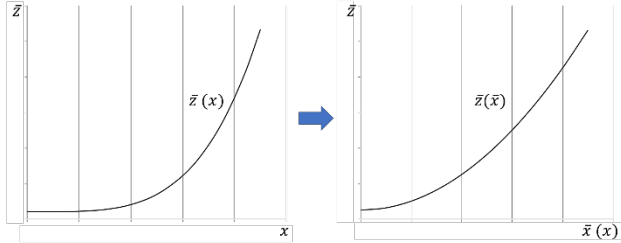


Figure 2. Illustration of coordinate transformation to enhance accuracy with equidistant nodes.

Chain rule must be used for calculation of derivatives. For example, if both the dependent and independent variables are transformed, i.e. $\bar{x} = \bar{x}(x)$ and $\bar{z} = \bar{z}(z, \bar{x})$, then the derivative in the i^{th} interval is given as:

$$\frac{dz_{\{i\}}}{dx} = \frac{d\bar{z}_{\{i\}}}{d\bar{x}} \left(\frac{\partial z}{\partial \bar{z}} \right)_{\bar{x}} \frac{d\bar{x}}{dx} \quad (4)$$

where

$$\frac{d\bar{z}_{\{i\}}}{d\bar{x}} = a_{i2} + 2a_{i3}(\bar{x} - \bar{x}_i) \quad (5)$$

2.2 Data Generation

The property (raw) data is generated using property models in the Air Conditioning Library, more specifically, short Helmholtz model for R134a and reference Helmholtz model for R1234yf. The data is fed to a Python script that solves for the spline coefficients. The coefficient data is then stored as MAT files for later use in the Modelica property model.

The phase boundary of the refrigerant is defined by a 1D spline $T(p_s)$. We have five 2D splines for different properties: temperature $T(p, h)$, density $\rho(p, h)$, entropy $s(p, h)$, dynamic viscosity $\mu(p, h)$, and thermal conductivity $\lambda(p, h)$. For consistency, the bubble and dew enthalpy are expressed as inverse of the 2D temperature spline: $h_{liq}^{INV}(p_s, T(p_s))$ and $h_{vap}^{INV}(p_s, T(p_s))$. Other properties like specific heat capacity can be derived from the 1D and 2D splines (Tummescheit, 2002; Thorade & Saadat, 2013).

We used 100 nodes for the 1D spline and about 120×120 nodes for the 2D splines. For 2D splines, global interpolation is performed for the whole (p, h) domain. To ensure the spline interpolation is accurate in the single phase region up to the phase boundary, the raw data was extrapolated from the single phase region into the 2-phase region. Furthermore, the pressure

nodes of the 1D spline $T(p_s)$ overlap with those of the 2D splines (up to the critical point).

2.3 Property Model Structure

The spline coefficient data in the MAT files is loaded once into the memory at initialization of the simulation, so the size of the data file only affects the CPU time at initialization but not during time integration. The structure of the model is shown in Figure 3. The top-level functions for different refrigerant properties are implemented in Modelica. These Modelica functions call the C functions (as external object) that evaluate, invert, and take derivative of the spline.



Figure 3. Structure of a thermodynamic property function call in the SBTL model in Modelica.

3 Verification and validation of SBTL for refrigerant property calculations

In this section, comparisons are made between a short formulation of Helmholtz energy EOS (short Helmholtz) model and the SBTL model for R134a. We first look at the CPU time of single function calls and then progress to a heat exchanger test bench and a full system model of vapor compression cycle in the Air Conditioning Library. Finally, we tested an SBTL model of R1234yf in several complex AC system models developed at the Ford Motor Company to evaluate its accuracy and performance in drive cycle simulations. The computer configuration for our tests is shown in Table 1.

Table 1. Configuration of the computer used for testing

| | |
|-------------------|------------------------------------|
| <i>Model</i> | Dell Precision M2800 Laptop |
| <i>Processor</i> | Intel® Core™ i7-4810MQ CPU |
| <i>RAM</i> | 16.0 GB |
| <i>System</i> | 64-bit, x64 based, Windows 10 Pro |
| <i>Software</i> | Dymola 2018 |
| <i>C compiler</i> | Visual Studio 2012 Express Edition |
| <i>Solver</i> | Euler (functions), Dassl(systems) |
| <i>Tolerance</i> | 1e-6 (to ensure mass conservation) |

3.1 Comparison of function call test results and performance

All the property functions of R134a were tested in the validity range and compared to the short Helmholtz model. Some of the results are listed in Table 2. In the dew enthalpy test, pressure ramped from 0.3 to 39.5

bar. For temperature and density derivative tests, the enthalpy ramped from 150kJ/kg to 500kJ/kg with pressure fixed at 0.3, 0.5, 1, 2, 5, 10, 20 and 39.5 bar. Each test ran for 1s with a fixed step size of 1e-4s, resulting in 1e4 evaluations. CPU time per evaluation was obtained by advanced profiling feature in Dymola.

Table 2. Comparisons of individual function calls

| Property | Relative error in % | CPU time short Helmholtz | CPU time SBTL | CPU Time reduced |
|---------------------------------------|---------------------|--------------------------|---------------|------------------|
| h_{vap} | <0.5% | 1.4e-6s | 7e-7s | 50% |
| T | <0.03% | >1.2e-5s | <2.0e-6s | >83.3% |
| $\frac{\partial \rho}{\partial h} _p$ | N/A ¹ | >2.8e-5s | <3.5e-6s | >87.5% |

Each row above represents a certain type of property functions. The dew enthalpy test is representative of saturation properties. The Helmholtz energy EOS uses a cubic spline while the SBTL method inverts the bi-quadratic spline of temperature to obtain saturation enthalpy. The temperature test shows the speed of calculating density, entropy, etc., i.e. evaluation of a 2D spline. The test for partial derivative of density demonstrates the speed-up of calculating partial derivatives, e.g. specific heat capacity, isobaric expansion coefficient, etc. The speed of temperature and the partial derivative functions varies because the computational costs for evaluation in the single-phase region and the two-phase region are different.

The SBTL method is significantly faster with only small deviations in the results. This is partially due to the use of lower order splines. Moreover, the Helmholtz energy EOS uses density and temperature as states, which requires iteration when calling property functions from pressure and enthalpy, while SBTL method simply evaluates spline or its derivatives.

A contour plot of percentage error in density spline evaluation (300×300 points) is shown in Figure 4. White color means the error is below 0.001%. The spline is very accurate in most of the region. Deviation from the reference value locates mainly in the surrounding of the critical point. The maximum error is about 2.4%, which appears inside the two-phase region close to the critical point.

¹Phase boundary locations are slightly different in the two models, making it hard to compare the results in terms of percentage deviation.

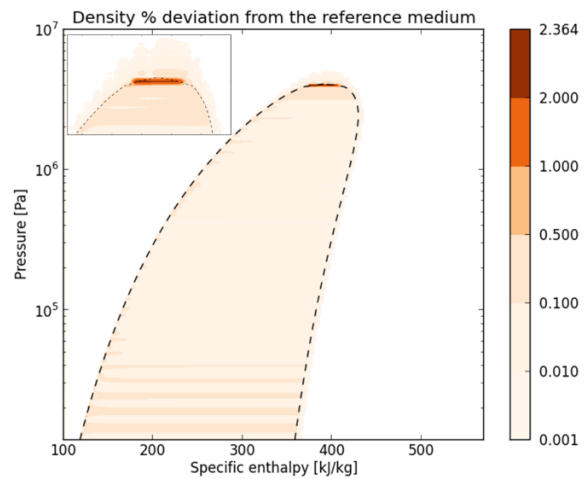


Figure 4. Contour plot of the % deviation of density. Enlarged plot in $p = 30\text{-}51$ bar, $h = 330\text{-}440$ kJ/kg.

3.2 Comparison of heat exchanger test results and performance

System models of vapor compression cycles usually consist of thousands of equations with complicated numerical structures produced by symbolic manipulations. Hence, further proof of concept, beyond property function tests, is required to evaluate the performance of the SBTL model in system level simulations. In a full cycle, the discretized heat exchangers usually have the highest number of property function calls and comprise a large part of the computational cost. Hence, a heat exchanger simulation is a great test case before jumping to a complete vapor compression cycle simulation. An evaporator test bench from the Air Conditioning Library, shown in Figure 5, is used for the test of the SBTL model for R134a. The test bench was simulated for 20s with a ramp in refrigerant mass flow rate increasing from 0.02 to 0.03 kg/s at $t = 5$ s to 7 s. Other boundary conditions are kept at constant.

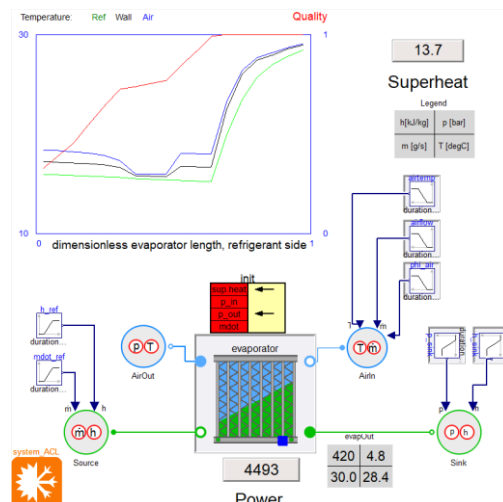


Figure 5. Evaporator test bench in ACL.

Comparisons of the cooling power and air outlet temperature between the SBTL model and the short Helmholtz model can be found in Figure 6. The air outlet temperatures from the two medium models overlap completely and the cooling powers only have small deviations.

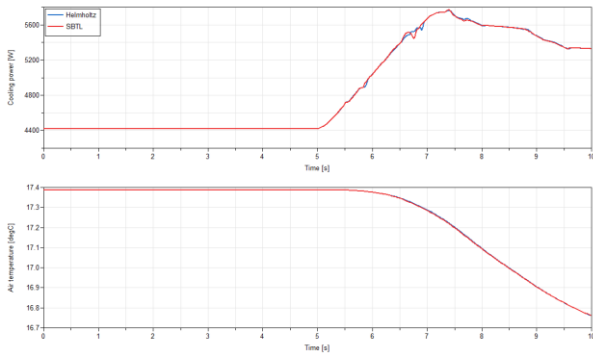


Figure 6. Comparison of cooling power and air outlet temperature. Blue – short Helmholtz, Red – SBTL.

Figure 7 shows the CPU time comparisons. The red curves are CPU time after initialization while the blue curves include CPU time of initialization of the model. Dotted curves are for short Helmholtz model while solid ones are for SBTL model. As we can see, the SBTL model was running twice the speed both at initialization and during the transient run.

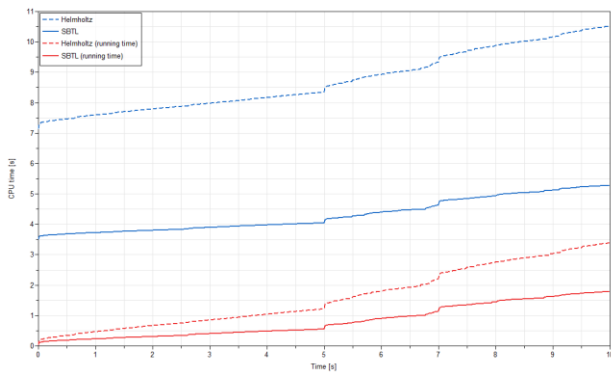


Figure 7. Comparison of CPU time. Dotted – short Helmholtz, Solid – SBTL; Red – CPU time after initialization, Blue – Total CPU time.

The heat exchanger test results demonstrate the speed-up provided the SBTL model beyond single function calls, and they serve as good indicators of the performance improvement in a full vapor compression cycle, as discuss in the later sections.

3.3 Comparison of system models in the Air Conditioning Library results and performance

The pull-down test from the Air Conditioning Library, depicted in Figure 8, is used to evaluate the performance of the SBTL model of R134a. The model is run for 4000s to get to a steady state. Results and

CPU time are benchmarked against the short Helmholtz R134a model.

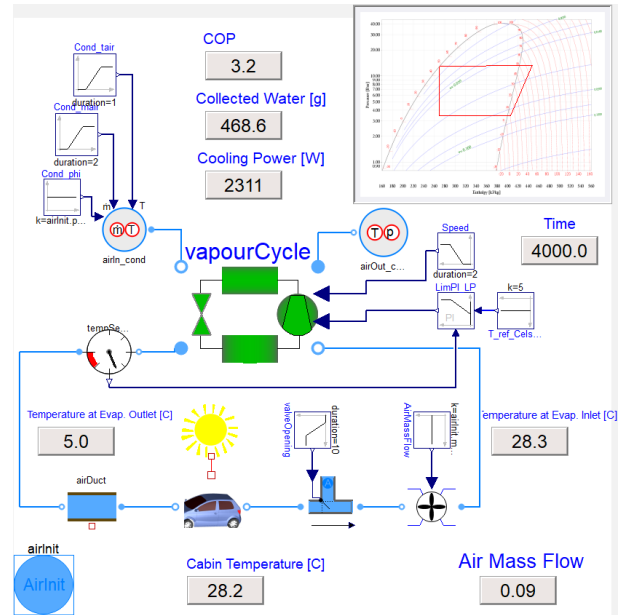


Figure 8. Pull-down test from ACL for an air conditioning cycle connection with vehicle cabin.

Deviations of some key results (trajectory during the whole simulation) from the short Helmholtz R134a model are listed in Table 3. The SBTL model replicated the result of the benchmark model very accurately.

Table 3. Deviations of key results in the pull-down test

| Key results | Deviation in % |
|----------------------------|----------------|
| Cooling Power | < 0.4 |
| Refrigerant mass flow rate | < 0.2 |
| Cabin temperature | < 0.002 |

The CPU time plots are shown in Figure 9. The upper one is for the entire 4000s simulation and the lower one zooms into the first 100s when most of the dynamics happened. The SBTL model reduced the CPU time by about 33%.

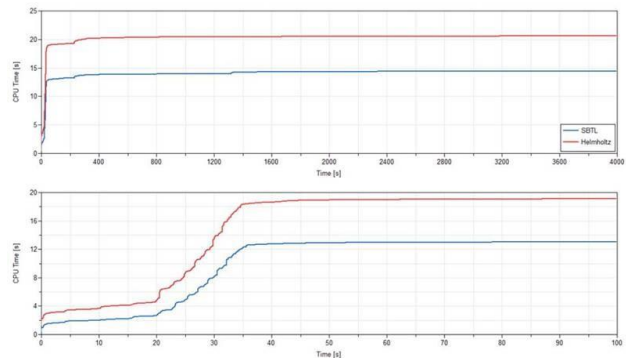


Figure 9. CPU time comparison of the pull-down test. Upper - entire simulations, Lower - first 100s.

All the tests discussed so far compare the SBTL model to the short Helmholtz model which is faster than reference state Helmholtz EOS. However, not every refrigerant in Air Conditioning Library has a short formulation. For example, R1234yf, proposed as a replacement for R1234a in automotive air conditioning systems, only has reference state Helmholtz EOS. Hence, we expected a larger performance improvement when using the SBTL method for R1234yf systems. The orifice cycle model from the Air Conditioning Library (Figure 10) was simulated for 180s to further study the performance improvement by SBTL model for R1234yf.

Table 4. Deviations of key results in the R1234yf orifice cycle simulations

| Key results | Deviation in % |
|----------------------------|----------------|
| Cooling Power | < 0.1 |
| Refrigerant mass flow rate | < 0.02 |
| Cabin temperature | < 0.001 |

The accuracy of the model is verified by comparing the dynamic trajectory of some key results, as listed in Table 4. As shown in the CPU time plot in Figure 12, the orifice cycle model with SBTL R1234yf ran twice as fast as the reference.

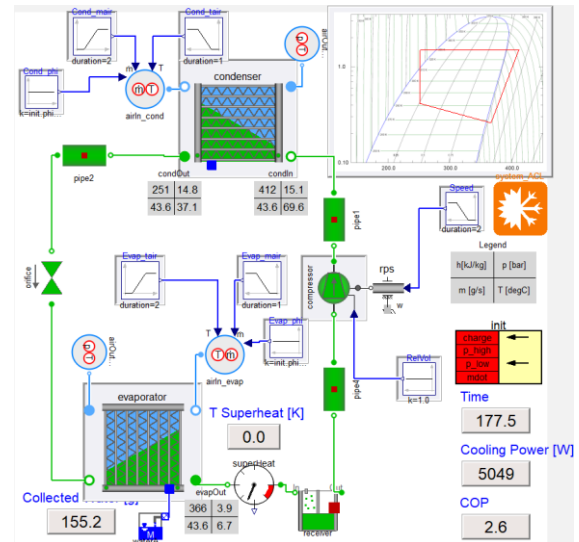


Figure 10. Orifice cycle model using R1234yf in ACL.

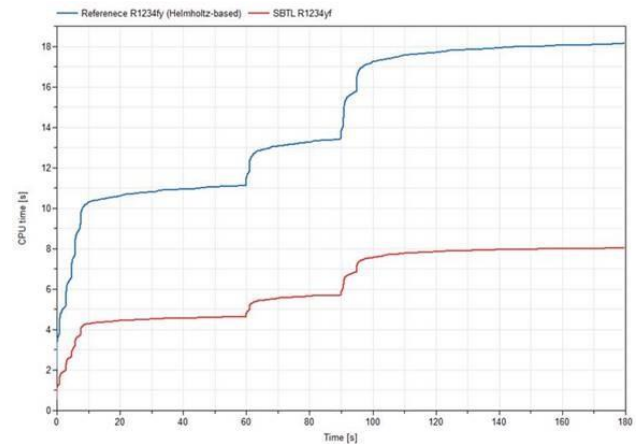


Figure 12. CPU time comparison of the orifice cycle simulations. Blue - Reference Helmholtz, Red – SBTL.

| # | Model | Check | Translate | Simulate | Translation Time | Initialization Time | Simulation Time | Test Specification | CPU time plot | Trajectory Check |
|------|----------------------------|-------|------------------|------------------|------------------|---------------------|-----------------|--------------------|---------------|------------------|
| 1 | Evaporator | pass | pass [Reference] | pass [Reference] | 15 | 0.6 | 1.6 (10.0) | Open [Reference] | View | pass |
| 2 | Compressor | pass | pass [Reference] | pass [Reference] | 8 | 0.0 | 0.1 (20.0) | Open [Reference] | View | pass |
| 3 | Condenser | pass | pass [Reference] | pass [Reference] | 13 | 0.2 | 1.3 (60.0) | Open [Reference] | View | pass |
| 4 | GasCooler | pass | pass [Reference] | pass [Reference] | 13 | 0.1 | 1.7 (60.0) | Open [Reference] | View | pass |
| 5 | PlateEvaporator | pass | pass [Reference] | pass [Reference] | 16 | 0.4 | 1.6 (10.0) | Open [Reference] | View | pass |
| 6.1 | IHX_1 | pass | pass [Reference] | pass [Reference] | 12 | 0.5 | 2.7 (100.0) | Open [Reference] | View | pass |
| 6.2 | IHX_2 | pass | pass [Reference] | pass [Reference] | 12 | 0.8 | 2.8 (100.0) | Open [Reference] | View | pass |
| 6.3 | IHX_3 | pass | pass [Reference] | pass [Reference] | 13 | 0.3 | 3.0 (100.0) | Open [Reference] | View | pass |
| 6.4 | IHX_4 | pass | pass [Reference] | pass [Reference] | 12 | 0.3 | 2.8 (100.0) | Open [Reference] | View | pass |
| 6.5 | IHX_5 | pass | pass [Reference] | pass [Reference] | 13 | 0.8 | 2.8 (100.0) | Open [Reference] | View | pass |
| 6.6 | IHX_6 | pass | pass [Reference] | pass [Reference] | 12 | 0.8 | 2.9 (100.0) | Open [Reference] | View | pass |
| 6.7 | IHX_7 | pass | pass [Reference] | pass [Reference] | 14 | 0.6 | 2.8 (100.0) | Open [Reference] | View | pass |
| 6.8 | IHX_8 | pass | pass [Reference] | pass [Reference] | 13 | 1.1 | 3.0 (100.0) | Open [Reference] | View | pass |
| 7 | SubcoolerCondenser | pass | pass [Reference] | pass [Reference] | 17 | 1.1 | 3.0 (20.0) | Open [Reference] | View | pass |
| 8 | TXVCycle_1 | pass | pass [Reference] | pass [Reference] | 21 | 2.0 | 9.3 (180.0) | Open [Reference] | View | pass |
| 8.1 | TXVCycle_2 | pass | pass [Reference] | pass [Reference] | 23 | 2.0 | 21.0 (180.0) | Open [Reference] | View | pass |
| 9 | OrificeCycle | pass | pass [Reference] | pass [Reference] | 18 | 1.2 | 9.5 (180.0) | Open [Reference] | View | pass |
| 10 | ChargeEstimation_1 | pass | pass [Reference] | pass [Reference] | 19 | 1.1 | 58.5 (100.0) | Open [Reference] | View | pass |
| 10.1 | ChargeEstimation_2 | pass | pass [Reference] | pass [Reference] | 21 | 1.1 | 61.2 (100.0) | Open [Reference] | View | pass |
| 11 | Co2Cycle | pass | pass [Reference] | pass [Reference] | 19 | 2.0 | 6.9 (180.0) | Open [Reference] | View | pass |
| 12 | TXVCycleOnOff | pass | pass [Reference] | pass [Reference] | 25 | 0.5 | 73.7 (150.0) | Open [Reference] | View | pass |
| 13 | Co2CycleOptimizedCOP | pass | pass [Reference] | pass [Reference] | 20 | 1.8 | 10.1 (200.0) | Open [Reference] | View | pass |
| 14 | CondReceiverIHXCycle | pass | pass [Reference] | pass [Reference] | 29 | 3.7 | 8.9 (200.0) | Open [Reference] | View | pass |
| 15 | InhomogeneousAirCondenser | pass | pass [Reference] | pass [Reference] | 54 | 0.4 | 2.5 (100.0) | Open [Reference] | View | pass |
| 16 | SuperheatControl | pass | pass [Reference] | pass [Reference] | 15 | 2.0 | 4.8 (200.0) | Open [Reference] | View | pass |
| 17 | TwinEvaporatorCycle | pass | pass [Reference] | pass [Reference] | 29 | 3.8 | 10.0 (180.0) | Open [Reference] | View | pass |
| 18 | SimulinkInterface | pass | pass [Reference] | pass [Reference] | 13 | 0.9 | 2.4 (100.0) | Open [Reference] | View | pass |
| 19 | InhomogeneousCSVAirSources | pass | pass [Reference] | pass [Reference] | 15 | 0.5 | 2.1 (100.0) | Open [Reference] | View | pass |

Figure 11. A part of the ACL regression test report. SBTL property model was used in the tests and the results were compared against reference results obtained by the Helmholtz property models.

3.4 Some comparisons from our full suite of the Air Conditioning Library regression tests

To ensure the robustness and accuracy of the SBTL property models, we used them in our existing regression tests of the Air Conditioning Library and compared to the reference results generated by the Helmholtz property model. A test was considered “pass” only if it compiled, simulated and produced results within a tight tolerance of the references. A small portion of the full regression test suite is shown in Figure 11.

3.5 Comparison of Ford AC system models results and performance

In this section, simulations are performed on two complex AC system models developed at Ford Motor Company R1234yf is used in both systems. Figure 13 depicts an AC system with two evaporators and one chiller connected in parallel in the loop. Simulations of the SC03 drive cycle were performed on this model. The cooling power of the evaporators and the chiller can be found in Figure 14. The SBTL model replicates the results from the Helmholtz model very closely. Figure 15 shows the comparison of CPU time. The SBTL model took only 509s to run, which is 85% of the real-time (598s), and it saves more than 60% of CPU time compared to the Helmholtz model.

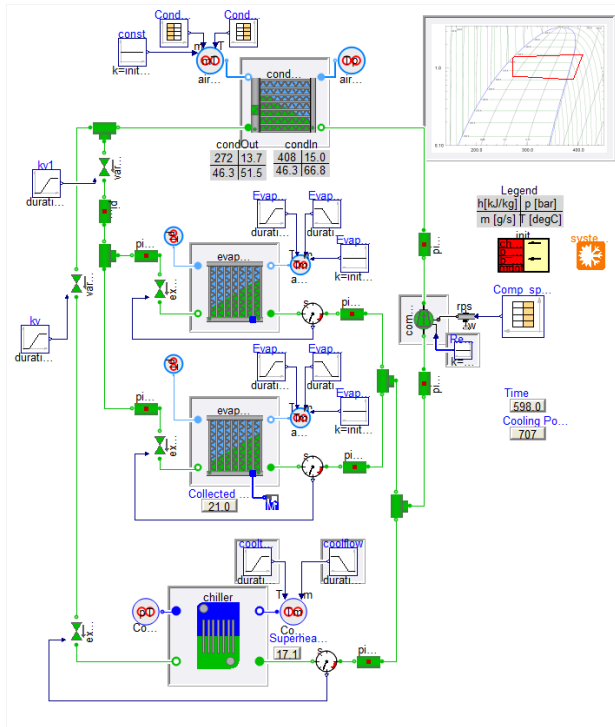


Figure 13. AC system with two evaporators and one chiller connected in parallel in the loop.

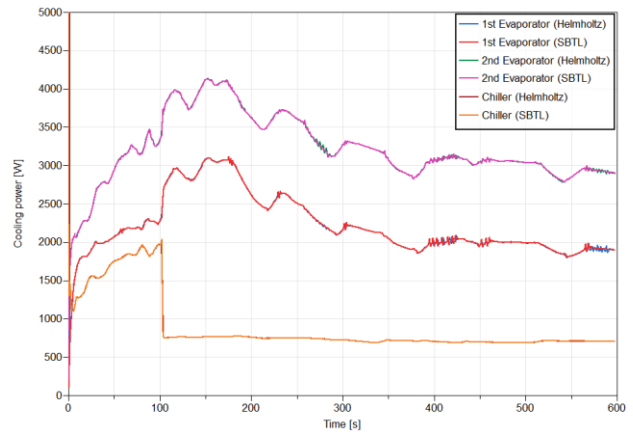


Figure 14. Cooling power of the evaporators and the chiller using Helmholtz property model and SBTL model.

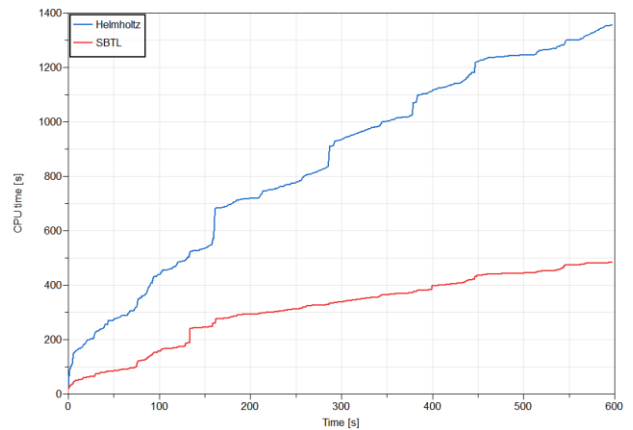


Figure 15. CPU time comparison of the AC system model shown in Figure 13.

Figure 16 is a vapor compression cycle with a chiller connected to a battery cooling loop. In the simulation, the compressor was off at $t = 0$ s, and the refrigerant loop was initialized with a certain mass flow rate, i.e. the simulation started at the moment when the compressor was turned off. The compressor was turned on again when the battery temperature (cooled by the cooling loop) is above a certain threshold. This shut down and startup test is challenging because of low refrigerant flowrate when the compressor is off and the fast dynamics when it restarts.

The compressor speed and refrigerant mass flow rate are plotted in Figure 17. The SBTL model predicts the mass flow well even during the fast transients after the compressor restarted. CPU time comparison can be found in Figure 18. The SBTL model saves more than 70% of the CPU time.

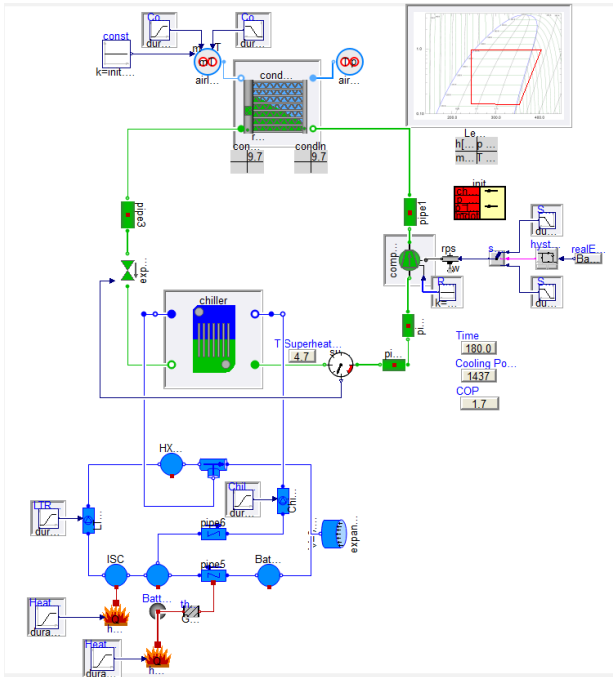


Figure 16. R1234yf vapor compression cycle with a chiller connected to a battery cooling loop.

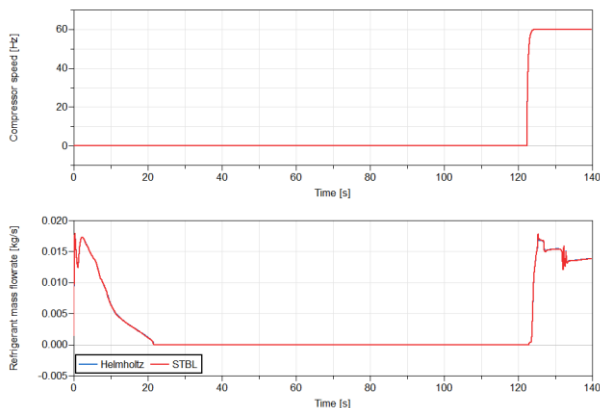


Figure 17. Compressor speed and refrigerant mass flow rate.

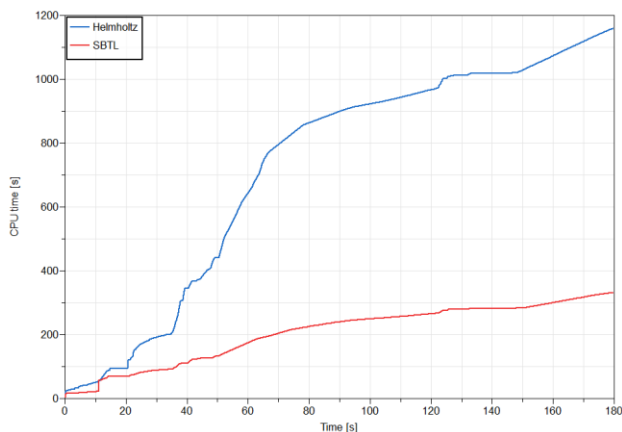


Figure 18. CPU time comparison for the model shown in Figure 16.

4 Conclusions

This paper summarizes an implementation of the SBTL method in Air Conditioning Library for fast calculation of refrigerant properties using Modelica language. The SBTL method, the data generation process, and the property model structure are explained. The SBTL refrigerant property models demonstrate significant improvement in computational speed in single function calls. In system simulations of AC cycle with R134a, the SBTL model cut the CPU time by 33% compared to the short Helmholtz model. The complex AC system models from Ford Motor Company run twice the speed with SBTL model of R1234yf than with reference Helmholtz model. The SBTL models for R134a and R1234yf will be available in the upcoming 2018.2 release (version 1.17) of Modelon's Air Conditioning Library.

References

- Tillner-Roth, R. and Baehr, H.D. An international standard formulation of the thermodynamic properties of 1,1,1,2-tetrafluoroethane (HFC-134a) for temperatures from 170 K to 455 K at pressures up to 70 Mpa. *Journal of Physical and Chemical Reference Data*, 23(5), 657-729, 1994.
- Span, R. and Wagner, W. Equations of state for technical applications. I. Simultaneously optimized functional forms for nonpolar and polar fluids. *International journal of thermophysics*, 24(1), 1-39, 2003.
- Richter, M., McLinden, M.O., and Lemmon, E. W. Thermodynamic Properties of 2, 3, 3, 3-Tetrafluoroprop-1-ene (R1234yf): Vapor Pressure and p–p–T Measurements and an Equation of State. *Journal of Chemical & Engineering Data*, 56(7), 3254-3264, 2011.
- Laughman, C., Zhao, Y., and Nikovski, D. Fast Refrigerant Property Calculations Using Interpolation-Based Methods. *International Refrigeration and Air Conditioning Conference*. Paper 1344, 2012.
- Schulze, C. W. A contribution to numerically efficient modelling of thermodynamic systems. *PhD Theses*, 2013.
- Aute, V. and Radermacher, R. Standardized Polynomials for Fast Evaluation of Refrigerant Thermophysical Properties. *International Refrigeration and Air Conditioning Conference*. Paper 1499, 2014.
- Kunick, M., and H. J. Kretzschmar. Guideline on the fast calculation of steam and water properties with the spline-based table look-up method (SBTL). *Technical report, The International Association for the Properties of Water and Steam*, Moscow, Russia, 2015.
- Späth, H. One dimensional spline interpolation algorithms. AK Peters/CRC Press, 1995.
- Späth, H. *Two dimensional spline interpolation algorithms*. AK Peters, Ltd., 1995.
- Tummescheit, H. Design and implementation of object-oriented model libraries using Modelica. *PhD Theses*, 2002
- Thorade, M. and Saadat, A. Partial derivatives of thermodynamic state properties for dynamic simulation. *Environmental earth sciences*, 70(8), pp.3497-3503, 2013.

Modelica-Based Dynamic Modeling of a Solar-Powered Ground Source Heat Pump System: A Preliminary Case Study

Defeng Qian¹ Zheng O'Neill¹

¹ Department of Mechanical Engineering, the University of Alabama, Tuscaloosa, AL, the United States
dqian@crimson.ua.edu, zoneill@eng.ua.edu

Abstract

This paper presents preliminary simulation results from a Modelica – based dynamic model of a solar-powered ground source heat pump (GSHP) system. The model is calibrated and tested using the data collected from a test rig in the laboratory. The preliminary data collected from the test rig includes temperatures for water and air loops, solar heat radiation flux, solar panel output power, and power consumption of the GSHP. This preliminary study is focusing on the comparisons between Modelica model outputs and experimental data, which includes the power consumption of the GSHP unit, unit cooling capacity, and unit coefficient of performance (COP).

Keywords: Ground source heat pump, Solar-Powered, Modelica

1 Introduction and Background

According to the recent research (Berardi, 2015), from 2007 to 2035, the global demand for oil will increase by 30%, while the demand for coal and natural gas will increase by 50%. Those data inform us that energy-related carbon emissions will increase significantly if there are no radical changes in the energy structure. According to the annual energy outlook report, the building sector consumed 40% of the energy and 70% of the electricity in the U.S. in 2017. (EIA, 2017). About 24% of all energy used in the nation was for space heating, cooling, and water heating in the buildings (DOE, 2010). Besides the system safety and occupants' thermal comfort, improving energy efficiency and reducing energy consumption in buildings is one of the most important priorities during the operation stages of buildings.

Enhancing building efficiency is one of the simplest, most immediate and most cost-effective ways to reduce the carbon emissions (Li and Colombier, 2009). In addition, integrating renewable energy sources into an efficient Heating, Ventilation, and Air-Conditioning (HVAC) would make a net-zero energy building (NZEB) possible. Theoretically, the goal of NZEB is to reduce the energy consumption (and demand) through efficient designs and operations, and utilize the renewable energy as a major energy source while the conventional energy sources play a backup role in the buildings. The key enablers for the NZEB include: 1) a

highly efficient building envelope, 2) high-performance HVAC systems with the advanced control strategy, and 3) a balance between the building energy consumption and onsite power generation (Besant, Dumont et al., 1979, Hayter, Torcellini et al., 2000, Marszal, Heiselberg et al., 2011, Marszal, Heiselberg et al., 2012, Attia, Hamdy et al., 2013).

1.1 Ground Source Heat Pump (GSHP)

As one of the most efficient systems on the market, the GSHP system has been proved as one of the most energy-efficient solutions for the building HVAC system for a wide variety of geology conditions (ASHRAE, 2007). GSHP system combines the heat pump and a ground loop heat exchanger for transferring the heat between the building and the ground source.

Compared to the air-source heat pump that utilizes the environmental air as the heat source/sink, the GSHP utilizes the earth such as groundwater or soil as the heat source/sink. Since ground maintains at a relatively constant temperature over the year, the GSHP system is about 45% more efficient than conventional air source heat pumps (EnergyStar, 2018).

In general, open-loop system, closed-loop system, and semi-open-loop system are the most common design for the GSHP system. According to Hutterer (Hutterer, 1997), an open-loop system uses the groundwater directly. The groundwater passes through the heat pump unit and is discharged back to the source. In a closed-loop system, the water or the water-antifreeze solution circulates in a continuous buried pipe, which acts as a ground heat exchanger between the ground source and the circulating fluid. Compared to the closed-loop system, an open-loop system is inexpensive and efficient; however, additional maintenance is required to prevent fouling of loops by organic matter, etc. In addition, discharge of water from an open loop system to a surface waterbody may require a permit. In the U.S., Environmental Protection Agency (EPA) requires reporting any injection of the water to a return well for groundwater heat pump systems. The semi-open-loop system (i.e., standing column well system) combines the advantage of both open-loop system and closed-loop system. The focus of this study is an open-loop GSHP system as shown in Figure 1. Due to the low pH value of the groundwater in the test location, a plate heat exchanger was introduced between the heat pump

condenser and the groundwater loop to avoid the potential corrosion of the condenser.

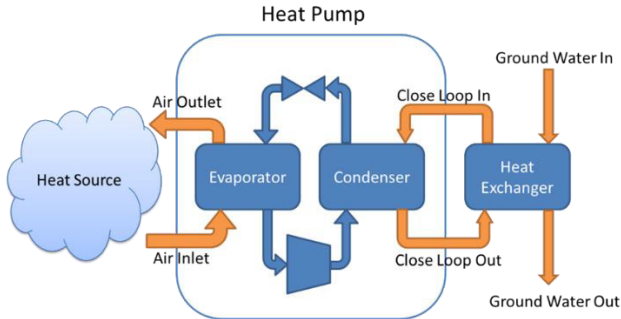


Figure 1. The open-loop GSHP system in this study

1.2 Renewable/Sustainable Energy (Solar Energy)

Growing population and technology evolutions caused the energy demand increased significantly in the past decades. Currently, as the primary source of the energy demand, fossil fuel shortage can be predicted while the energy demand is increasing continuously (Pérez-Lombard, Ortiz et al., 2008, Shafiee and Topal, 2009). The fossil fuel is a type of finite resource and responsible for significant carbon emissions (Ediger, Hoşgör et al., 2007). Back in 2007, the World Energy Outlook predicted that 84% of the energy demand would depend on the fossil fuels in 2030 (Shafiee and Topal, 2009). This type of situation motivates people to explore an alternative way to satisfy the energy demand.

Meanwhile, renewable energy can be obtained from the natural sources, including solar, wind, biomass, etc. It is considered as an unlimited and environment-friendly energy source (Twidell and Weir, 2015). As one of the most widely used renewable sources, solar energy has been widely used for solar thermal and solar power applications. It is not only a promising source but also abundant energy. Kannan and Vakeesan's recently did a review study about the solar energy and its future, and their study listed the situation, potential applications, and barriers to the solar industry. The study was a valuable reference for solar-related manufacturers, researchers, and decision-makers to take further actions (Kannan and Vakeesan, 2016). Figure 2 shows a map of global horizontal irradiation (GHI). The map shows the potential solar energy is at a range of 1,500 to 2,200 kWh/m² in the United States.

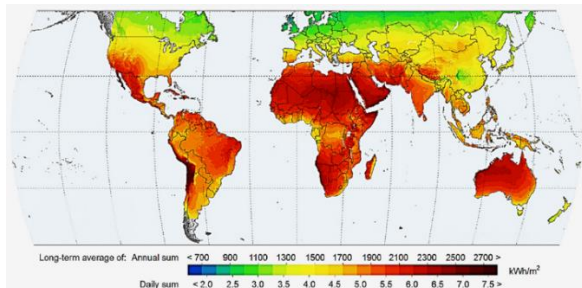


Figure 2. Maps of global horizontal irradiation (GHI) (Kannan and Vakeesan, 2016)

1.3 Modeling with Modelica

As the mathematical modeling and simulation became the key factors in engineering, computational tools were developed to satisfy the needs of efficient engineering. Modelica-based models (Modelica, 2018) compiled using Dymola (Dymola, 2018) are used in this study. Modelica is an equation-based and object-oriented modeling language for complex multi-physics systems. The use of Modelica for the built environment is promising as buildings involve multiple physical phenomena (e.g., heat transfer, fluid dynamics, electricity, etc.) and are complex in terms of their dynamics (e.g., the coupling of continuous time physics with discrete time and discrete event control). In addition, the problem size can be varied from equipment to buildings and communities with electrical distribution grids. An advantage of Modelica is the modularity of the language that allows modification of the code according to the specific needs of the application. The object-orientation enables extension and reuse of components, and the use of standardized interfaces enables collaboration across physical domains and disparate developer groups. Modelica has been used to model the complex physical system, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power systems or process-oriented subcomponents (Modelica, 2018).

In general, a high-performance component and system could be less efficient without the appropriated and robust control strategies. Model-based control has been widely used in automobile, aerospace, and industry processes, and starts to emerge in the building industry. The dynamic modeling capability offered by Modelica provides a good framework for such model-based control design.

In this study, a Modelica-based dynamic model is developed to simulate the dynamics of solar-powered GSHP system. The commercial Modelica library – Vapor Cycle library was adopted (Modelon, 2018) for steady-state and transient simulation of a refrigeration cycle in the ground source heat pump. It is compatible with some Modelica libraries such as Liquid Cooling Library, Heat Exchanger Library, etc. (Modelon, 2018).

This paper presents the simulation results from the Modelica model of the GSHP unit, together with comparisons between simulation results and measurements from a test rig. The performance analysis and comparisons were conducted for the cooling mode only since the test rig was only operated in such mode. Only modeling and validations of the heat pump component of the test rig are included in this paper. The performance comparison covers the power consumption of the heat pump unit, unit cooling capacity, and unit COP.

2 Methodology

2.1 Test Rig Physical System Setup

As shown in Figure 3, a solar-powered GSHP system was the focus of this research to explore the benefits of integrating GSHP and the solar panels. In this test rig, a ¼-ton water-to-air GSHP is connected to two 60-foot deep wells. A group of Solar PV panels of 1.12 kW is connected to two 800 Ah battery banks, which are used to power the GSHP system and a 270 Watts DC powered well pump. During the daytime, solar PV panels convert solar photons into electrical energy, which will be stored in battery banks. Whenever the system is on demand, the battery banks can provide the electrical power. Meanwhile, a comprehensive performance monitoring and data acquisition system is installed. Figure 4 shows the GSHP unit with measurement points. Table 1 lists the major sensors deployed in the test rig. A three-hour testing data is used in this study.

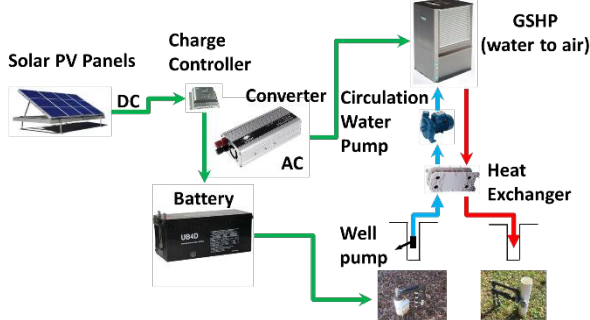


Figure 3. Overall system schematics of the studied solar-powered GSHP system

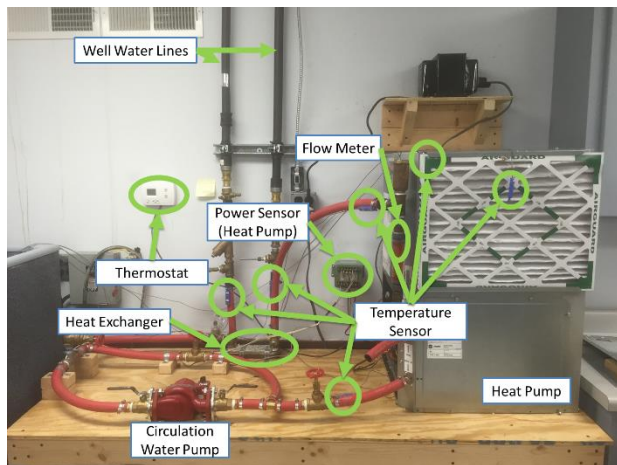


Figure 4. The heat pump unit with measurement points

A comprehensive performance monitoring and data acquisition system was installed (Figure 5). The data acquisition is combined with a chassis system which has five hybrid slots, three PXI Express slots (up to 250MB/s per-slot bandwidth and 1.75 GB/s system bandwidth) and two input modules. The thermocouple

input module has 32-channel, eight built-in cold junction compensation channels and a 0.3 °C accuracy. The voltage module has 16 analog inputs, two analog outputs, 16-bit resolution and a range of +10V.

Table 1. Major Sensors Deployed in the Test Rig

| Position | Sensor* |
|--|----------------------|
| Ground Water Inlet (from the well) | Temperature Sensor 1 |
| Ground Water Outlet (to the well) | Temperature Sensor 2 |
| Circulate Water Inlet (to the GSHP) | Temperature Sensor 3 |
| Circulate Water Outlet (from the GSHP) | Temperature Sensor 4 |
| Air Intake (to the GSHP) | Temperature Sensor 5 |
| Air Outlet (from the GSHP) | Temperature Sensor 6 |
| Power Sensor (GSHP) | Power Transducer |
| Solar Panel Input Voltage | Voltage Divider |
| Solar Panel Input Current | Current Shunt |

*The temperature sensor is a T-type thermocouple.

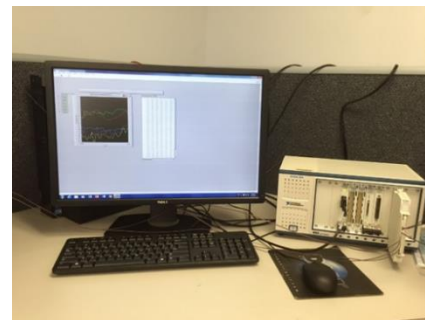


Figure 5. Data acquisition system in the test rig

2.1.1 Supply Side

The supply side for the studies system includes solar panels, battery banks, and a charge controller. Two sets of solar panels were used to charge the battery banks. The first set has 20 panels in two groups, and 16 of them were connected in four series loops (48 Volt). Each panel has a rated 50 Watts output. Thus, the total output from 16 panels would be 800 Watts. The second set has four panels with a rated output of 80 Watts per panel. The second set has a maximum output 320 Watts. However, since the first set was installed more than 20 years ago, a certain amount of performance degradation can be expected for these solar panels, while the second set is expected to operate close to the rated condition. Two battery banks are introduced as the energy storage devices, as one battery bank serves the GSHP, another one provides electricity to the well pumps in the system. Whenever the GSHP system is on demand, the battery banks will provide the electricity to maintain the system in operation.

2.1.2 Demand Side

The demand side of the studied system includes a GSHP unit, a circulation pump, and well pumps. The heat pump adopted in this study has a rotary type compressor, rated cooling capacity is 2,638 Watts. The unit uses R-410a as the refrigerant, with charging weight of 0.751 Kg. Under the normal operating condition, the refrigerant pressure is 3,103 kPa at the condenser side, and 1,724 kPa at the evaporator side. For the air side, the maximum external static pressure is 17,436 Pa. The rated operating voltage is 208 Volts, and the short-circuit current rating is 5 kA at 600V.

2.2 Modelica Model

A Modelica – based dynamic model is used to study the behaviors of the GSHP system. Only the modeling of the GSHP unit is included in this paper. The GSHP unit component (e.g., compressor, condenser, evaporator, expansion valve) were modeled using the existing Modelica library-Vapor Cycle Library (Modelon, 2018). The initial conditions were adjusted based on the test conditions and the manufacturer’s specifications. The working fluid in the studied GSHP is R410a, and the secondary side fluid is water at the condenser side and air at the evaporator side. The well is not directly modeled in this paper, heat source and heat sink with constant temperatures was specified using the data from measurements. More details of this assumption are provided in the next section.

Simulation models were created and compiled by using the multi-engineering dynamic simulation tool Dymola (version 2017 FD01) (Dymola, 2018). Table 2 lists the major components used in this model.

Figure 6 shows the Dymola model of the studied GSHP unit. System status data such as the power consumption of the compressor, heating/cooling rate, and the COP are pulled out and can be directly read from the main panel.

Table 2. Major Components in the Modelica model

| Component | Model Descriptions |
|-----------------|--|
| Condenser | Heat exchanger; Counterflow; R410a as working fluid; Water as liquid |
| Evaporator | Heat exchanger; Counterflow; R410a as working fluid; Air as liquid |
| Expansion Valve | Simplified Thermal Expansion Valve model, based on compressible flow valve in IEC 534/ISA S.75 standards |
| Compressor | Fixed displacement compressor with speed and pressure ratio dependency |

| | |
|---------------|---|
| Liquid Source | Modelon.Media.PreDefined.Liquids.IncompressibleWater is the Medium on condenser side; VaporCycle.Media.Air.Moist AirNoFreezing is the medium on evaporator side. |
| Liquid Sink | Modelon.Media.PreDefined.Liquids.IncompressibleWater is the Medium on condenser side; VaporCycle.Media.Air.Moist AirNoFreezing is the medium on evaporator side. |

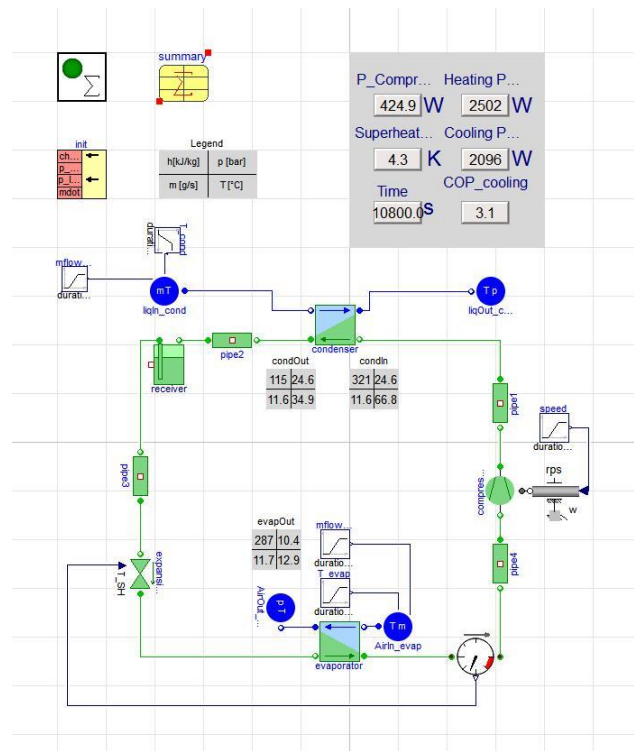


Figure 6. A Dymola model of the GSHP unit

2.2.1 Assumptions of the current Modelica model of GSHP unit

In this preliminary study, the following assumptions are used:

1. Constant temperatures were used in the Modelica model for the heat sink (i.e., the water inlet temperature on the condenser side) and the heat source (i.e., the air inlet temperature on the evaporator side). This assumption is a close approximation as observed from the actual testing data. Figure 7 shows that both the water inlet temperature and the air inlet temperature were relatively constant after the system went into the steady state (roughly 20 °C and 22 °C respectively).
2. The current Modelica model didn't include the blower fan, which is another major component

of the GSHP unit in the test rig. The current test rig only measured the total power consumption of the GSHP unit, which is the sum of the power consumption of the compressor and the fan. To have a fair comparison between the measurements and the model predictions, in this paper, a constant fan power consumption was assumed to get the total power consumption of the GSHP unit from the Modelica model.

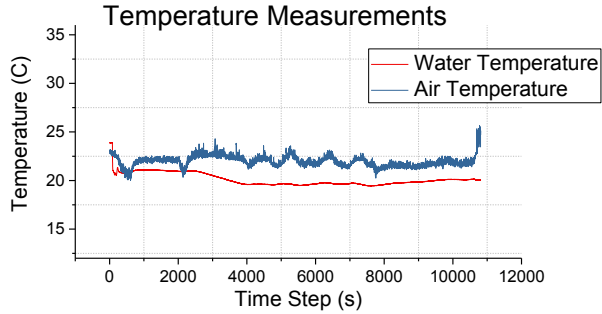


Figure 7. Water inlet and air inlet temperatures from measurements

2.3 System Performance Analysis

2.3.1 The Coefficient of Performance (COP)

The COP is a dimensionless parameter which is used to measure the efficiency of the heat pump. A higher value of COP corresponds to the better performance of the heat pump. In this study, the COP of the GSHP unit was calculated by using the following equation:

$$COP_{HP} = \frac{q_{cooling}}{P_{HP}} \quad (1)$$

Where COP_{HP} is the COP of the heat pump; $q_{cooling}$ is the cooling capacity (W); P_{HP} is the power consumption of the heat pump (W).

In a heat pump unit, there are multiple components consumes electrical energy during the operation. This study assumes only the compressor and the blower fan consume the energy, while the energy consumptions from other components are negligible during the operation. Therefore, the power consumption for the GSHP unit is defined as follow:

$$P_{HP} = P_{compressor} + P_{fan} \quad (2)$$

Where $P_{compressor}$ is the power consumption of the compressor (W); P_{fan} is the rated power for the blower fan (W). Currently, only P_{HP} is measured in the test rig.

2.3.2 Heat Pump Energy Balance

The energy balance on the heat pump represents the system performance in either the cooling or heating mode. The energy balance is based on the energy consumption of the heat pump, the heat exchange rate between the water-side and airside of the heat pump, and the cooling or heating rate of the heat pump. In this paper, all the tests were conducted for the cooling mode. The following equation is used to calculate the cooling

rate and check the total energy balance (Qian, Niu et al., 2016):

$$P_{HP} + q_{rej} + q_{cooling} = 0 \quad (3)$$

Where q_{rej} is the heat rejection rate (W) from the heat pump to the water loop; $q_{cooling}$ is the cooling capacity (W) of the GSHP unit.

2.3.3 Heat Rejection

The heat rejection of a water-to-air heat pump counts the energy transfer from the air to the water loop. In this study, the heat rejection rate was calculated according to the flow rate of the circulation groundwater and temperature difference of the circulation groundwater loop. Equation (4) is used to calculate the heat rejection of the GSHP unit in this study:

$$q_{rej} = \dot{m}_{water} \times C_p \times \Delta T_{CL} \quad (4)$$

Where \dot{m}_{water} is the mass flow rate (kg/s) of the circulating groundwater; C_p is the specific heat of water (J/kg-K); ΔT_{CL} is the temperature difference (K) between the condenser water inlet and outlet of the heat pump unit. This temperature difference is measured in the test rig.

2.3.4 Measure of Goodness

Statistic performance metrics help to determine how well a model can predict the performance of the system compared to the measurements. In this study, the coefficient of variation of the root mean square error (CVRMSE) and normalized mean bias error (NMBE) were used to determine the model accuracy (ASHRAE, 2002).

Root mean square error (RMSE) is a frequently used metric to measure the errors between model predictions and actual measurements, s shown in Equation 5:

$$RMSE = \sqrt{\frac{\sum (y_{act} - y_{mod})^2}{(n - p)}} \quad (5)$$

CVRMSE is the coefficient of variation of the root mean square error, as shown in Equation 6.

$$CVRMSE = \frac{RMSE}{\bar{y}_{act}} \times 100\% \quad (6)$$

NMBE is the ratio of the differences between actual measurements and simulated results to the degrees of freedom and mean value of the actual measurement.

$$NMBE = \frac{\sum (y_{act} - y_{mod})}{(n - p) \times \bar{y}_{act}} \times 100\% \quad (7)$$

Where y_{act} and y_{mod} are the actual measured and model predicted result; \bar{y}_{act} is the average of the actual measured data; n is the number of observation; p is the number of parameters in the regression model.

3 Results and Discussions

After the Modelica model was developed, a data set which was collected from the test rig was used to test

and evaluate the accuracy of the model. The Modelica model was built and simulated under the same operating conditions of the testing. The testing condition was as follows:

- 1) The comparisons between the actual measurements and simulation results were based on a three-hour testing.
- 2) The GSHP system was operating in a cooling mode.
- 3) The environment temperature was controlled at 22 °C, while the discharge air temperature set point of the GSHP system was 16 °C. The GSHP system maintained in operation during the three-hour test.
- 4) The flow rate of circulation groundwater was measured at 0.454 m³/hr, while the air flow rate was maintained at 0.134 m³/sec.
- 5) The measured temperature values include inlet water temperature at the condenser side and inlet air temperature at the evaporators side was used as the inputs to the Modelica model.
- 6) Other settings were referred to the manufacturer's specifications.

3.1 Power Consumption of the GSHP Unit

In the testing setup, the compressor is one of the major energy consumers in the system, the energy consumption of the compressor determines the performance of the GSHP system. Meanwhile, the testing unit was operating as a fixed air flow rate, the power consumption of the blower fan was assumed to operate in a rated condition (248 Watts) in this paper for the simplicity.

Figure 8 shows a comparison of the unit power comparison. After the system reached a steady state, the actual measurement was about 600 Watts, while the simulated power consumption of the unit was averaged at 670 Watts. The difference for the average power consumption during the three-hour testing period was approximately 11.67%.

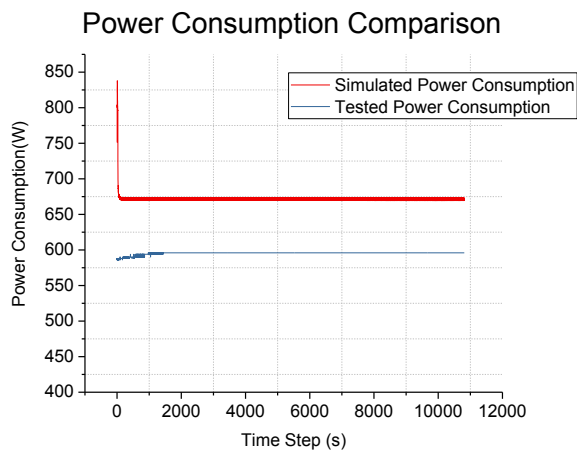


Figure 8. Comparisons of power consumption of the GSHP unit

As mentioned in section 2.2.1, a constant fan power consumption was assumed to get the total power consumption of the GHSP unit in the Modelica model since the fan was not included in the model. However, the fan power consumption most likely will not be maintained at the rated condition during the testing. This could explain why the simulated unit power consumption was consistently larger than the measurements. A fan model will be included into the Modelica model, and an additional measurement point for actual fan power consumption will be added as well.

3.2 Cooling Capacity

As mentioned in the previous section, the testing unit was a ¾-ton water-to-air GSHP system, which has a rated cooling capacity at 2,638 Watts. The actual cooling capacity was calculated by using Equation 3 in this paper.

Figure 9 shows the comparisons of the measured and simulated cooling capacity. According to the testing result, the actual cooling rate was maintained around 2,300 Watts after the system went into a steady state. As the simulation result was around 2,100 Watts, there was roughly 8.69% difference for the average cooling capacity between the test measurement and the simulation result.

In the current test rig, although the indoor air temperature was controlled and measured, the humidity in the lab was not controlled and measured. The moist air medium used in the Modelica model probably has the different humidity ratio compared to actual conditions in the test rig. This certainly would cause a different latent load between the simulation and the test cases. Theoretically speaking, more moist air could lead to a larger latent load for the heat pump unit, then result in a higher cooling capacity.

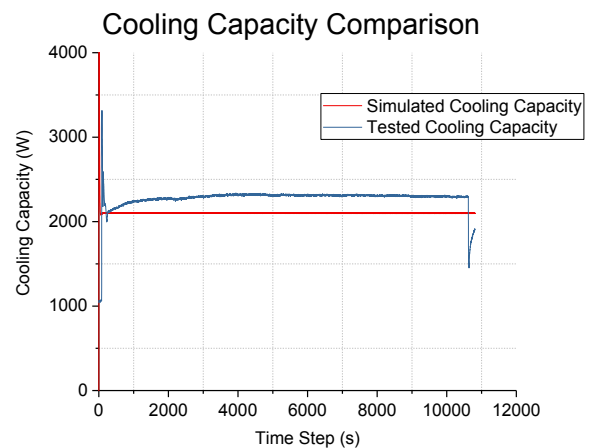


Figure 9. Comparisons of the GSHP unit cooling capacity

3.3 COP

The testing rig was operating under the cooling mode, only the cooling COP is discussed in this paper. The cooling COP from the testing was calculated using Equation 1.

Figure 10 shows the comparisons of the cooling COP for the three-hour testing period. After the test went into a steady state, the measured COP was about 3.8, while the simulation result was around 3.2. The comparison indicated 15.79% difference on mean on the COP.

As shown in the previous sections, actual test measurements indicate a higher cooling capacity and a lower power consumption. Thus, the COP from the testing was higher compared to the simulated COP from the Modelica model.

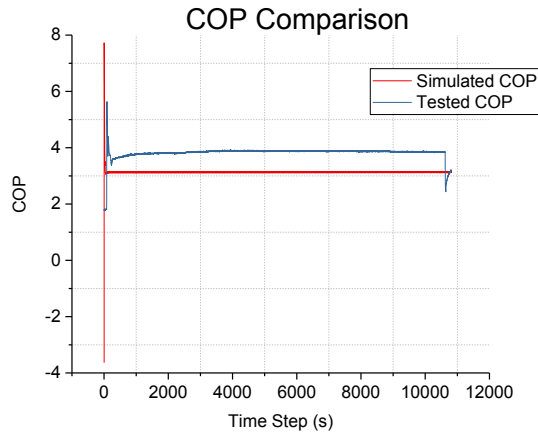


Figure 10. Cooling COP during the simulation.

3.4 Model Evaluation Metrics

Table 3 shows the measures of goodness for the model accuracy in this study. The CVRMSE were around 12.29%, 12.92%, and 19.69% for power consumption, cooling capacity, and COP respectively. The NMBE for these three outputs was 12.85%, 7.42%, and 17.98% respectively.

Table 3. Performance evaluation of the Modelica model

| Category | CVRMSE | NMBE |
|-------------------|--------|--------|
| Power Consumption | 12.92% | 12.86% |
| Cooling Capacity | 12.29% | 7.42% |
| COP | 19.69% | 17.99% |

4 Conclusions and Future Work

This study presents the preliminary results from the Modelica-based modeling of a GSHP unit. The model predictions were compared with measurements from the test rig. The current Modelica-based model can simulate the performances of the GSHP unit. The output trends for the Modelica simulation match with those from measurements well.

This Modelica model will be extended to a full scale of the solar-powered GSHP system that includes solar panels, battery banks, charge controller, and groundwater wells. Some of the ongoing and future work are listed as follows:

- 1) Modelica model of the supply side of solar panels, which will use the actual weather data as the

input to estimate the power generation of the solar panels.

- 2) On the groundwater side, a Modelica model of the groundwater well will be developed so the impact of ground (e.g., thermal conductivity and hydraulic conductivity) can be further analyzed.
- 3) A comprehensive system model of solar-powered ground source heat pump system will be validated using the measurement from the test rig. Testing data will be collected for a longer period for both heating and cooling modes.
- 4) Dynamic inputs such as weather information, room air and groundwater temperature profiles will be used as inputs in the full scaled model to study the dynamic performance of the system.
- 5) After the system model is developed and validated, this dynamic model will be used for the following applications:
 - Model-based control of the solar power generation system and ground source heat pump system, and the combination of these systems for a better building to grid integration.
 - Local heat pump controller design using this dynamic model in the Hardware-in-the-loop testing.

Nomenclature

ASHRAE: American Society of Heating, Refrigerating and Air-Conditioning Engineers
COP: Coefficient of Performance
CVRMSE: coefficient of variation of the root mean square error
EPA: Environmental Protection Agency
GHI: global horizontal irradiation
GSHP: Ground Source Heat Pump
HVAC: Heating, Ventilation, and Air-Conditioning
NZEB: Net Zero Energy Building
NMBE: normalized mean bias error
RMSE: Root mean square error

References

ASHRAE (2002). "ASHRAE guideline 14-2002. Measurement of energy and demand savings." American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc., Atlanta, GA.

ASHRAE (2007). "HVAC Applications Handbook." American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc., Atlanta, GA.

Attia, S., M. Hamdy, W. O'Brien and S. Carlucci (2013). "Assessing gaps and needs for integrating building performance optimization tools in net zero energy buildings design." *Energy and Buildings*, **60**: 110-124.

Berardi, U. (2015). "Building energy consumption in US, EU, and BRIC countries." *Procedia Engineering*, **118**: 128-136.

Besant, R. W., R. S. Dumont and G. Schoenau (1979). "The Saskatchewan conservation house: some preliminary performance results." *Energy and Buildings*, **2**(2): 163-174.

- DOE (2010). Buildings Energy Data Book, 2011.
- Dymola (2018). "DYMOLA Systems Engineering." from <https://www.3ds.com/products-services/catia/products/dymola/>.
- Ediger, V. Ş., E. Hoşgör, A. N. Sürmeli and H. Tatlıdil (2007). "Fossil fuel sustainability index: An application of resource management." *Energy Policy*, **35**(5): 2969-2977.
- EIA (2017). Annual Energy Outlook, U.S. Energy Information Administration. U. S. Department of Energy.
- EnergyStar (2018). "Geothermal Heat Pumps." from https://www.energystar.gov/products/heating_cooling/heat_pumps_geothermal.
- Hayter, S., P. Torcellini, R. B. Hayter and R. Judkoff (2001). The energy design process for designing and constructing high-performance buildings, Clima 2000/Napoli 2001 World Congress.
- Huttrer, G. W. (1997). "Geothermal heat pumps: an increasingly successful technology." *Renewable Energy*, **10**(2-3): 481-488.
- Kannan, N. and D. Vakeesan (2016). "Solar energy for future world:-A review." *Renewable and Sustainable Energy Reviews*, **62**: 1092-1105.
- Li, J. and M. Colombier (2009). "Managing carbon emissions in China through building energy efficiency." *Journal of Environmental Management*, **90**(8): 2436-2447.
- Marszal, A. J., P. Heiselberg, J. S. Bourrelle, E. Musall, K. Voss, I. Sartori and A. Napolitano (2011). "Zero Energy Building—A review of definitions and calculation methodologies." *Energy and Buildings*, **43**(4): 971-979.
- Marszal, A. J., P. Heiselberg, R. L. Jensen and J. Nørgaard (2012). "On-site or off-site renewable energy supply options? Life cycle cost analysis of a Net Zero Energy Building in Denmark." *Renewable Energy*, **44**: 154-165.
- Modelica (2018). "Modelica and the Modelica Association." from <https://www.modelica.org/>.
- Modelon (2018). "Vapor Cycle Library." from <http://www.modelon.com/products/modelon-library-suite/vapor-cycle-library>.
- Pérez-Lombard, L., J. Ortiz and C. Pout (2008). "A review on buildings energy consumption information." *Energy and Buildings*, **40**(3): 394-398.
- Qian, D., F. Niu, S. Kavanaugh and Z. O'Neill (2016). "Investigation on A Ground Source Heat Pump System Integrated With Renewable Sources." International Compressor Engineering, Refrigeration and Air Conditioning, and High Performance Buildings Conferences. West Lafayette, IN. July 11 - 14, 2016.
- Shafiee, S. and E. Topal (2009). "When will fossil fuel reserves be diminished?" *Energy Policy* **37**(1): 181-189.
- Twidell, J. and T. Weir (2015). Renewable energy resources, Routledge.

Coalesced Gas Turbine and Power System Modeling and Simulation using Modelica

Miguel Aguilera¹ Luigi Vanfretti² Tetiana Bogodorova³ Francisco Gómez⁴

¹Instituto Costarricense de Electricidad (ICE), Costa Rica, maguilera@ice.go.cr

²Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute (RPI), USA, vanf1@rpi.edu

³Ukrainian Catholic University, Faculty of Applied Sciences, Lviv, Ukraine, tetiana.bogodorova@gmail.com

⁴KTH Royal Institute of Technology, Stockholm, Sweden, fragom@kth.se

Abstract

This work reports how the multi-domain physical modeling and simulation Modelica language has been employed to create a benchmark power grid and gas turbine model within the ITEA3 OpenCPS project. The modeling approach is not only shown to be useful to test the functionalities of the OpenCPS toolchains, but it also could give rise to potential applications in power system domain studies where the widely-accepted turbine-governor models are not rich enough to represent the multi-domain system dynamics.

Keywords: Gas turbine modeling, Modelica, Multi-domain modeling and simulation, Power systems, OpenIPSL, ThermoPower

1 Introduction

1.1 Motivation

Variable energy resources, like wind and solar power, require a special attention due to the challenges that its intermittent nature poses to the power grid operation. To safely integrate these energy sources to power systems, acceptable levels of reliability and security and affordable prices are required (Carnegie Mellon University, 2013).

The operational flexibility of gas power plants makes them a good complement to variable renewable sources. It is very likely that policies will promote the increase of gas power, at least in the next decade, especially because they produce less emissions than coal power plants (IEA, 2016).

The variability of wind and solar power can be expressed as slow or fast fluctuations. Both, the less environment-friendly coal power plants and, combined cycle gas plants can be used to compensate slow power intermittency. On the other hand, power increase/reduction required to deal with fast power fluctuations can be achieved by means of fast response sources like gas natural turbines (Carnegie Mellon University, 2013).

Resilient operation of power systems with high penetration of variable energy resources (VERs)

depends, among other factors, on more trustable forecasts and accurate models that can be tailored to the several kinds of power system simulations and analysis. Existing gas turbine models, such as GGOV1, IEEE (De Mello & Ahner, 1994) and Rowen (Rowen, 1983, 1992), have different levels of complexity and accuracy. Simplicity was a desired property for the first proposed models, primarily due to computer power and turbine modeling data availability limitations of the time when they were proposed, the 1980s or early 1990s (De Mello & Ahner, 1994; Hannett & Khan, 1993). Such was the case of the GAST model which was widely used in the United States, but was demonstrated to be inaccurate and thus replaced by the somewhat more complex GGOV1 model (Pereira, Undrill, Kosterev, Davies, & Patterson, 2003). The widely-accepted models GGOV1, IEEE and Rowen do not employ a detailed physical representation of the gas turbine dynamics; instead, they model dynamics using abstractions in the form of logic and transfer functions which results in loss of information of non-linear physical dynamics. In fact, it has been recently shown (Yee, Milanovic, & Hughes, 2008) that more detailed models are required to include the grid frequency dependency behavior of gas turbines with the aim of undertaking power system stability studies when the turbines are exposed to abnormal system frequency behavior (e.g. black start, islanded operation, etc.). On the other hand, the correctness of the more complex physical models of gas turbines relies on the availability of turbine modeling data from the manufacturers, who create and then share such models with turbine owners, but are rarely available to most grid analysts due to IP concerns (Yee et al., 2008).

The CEN-CENELEC-ETSI Group recommends the use of the IEC CIM (Common Information Model) for information exchange, which has been mandated at the EU level (CEN-CENELEC-ETSI Smart Grid Coordination Group, 2012). The information exchange required to meet the needs of coordination of transmission system operators (TSOs) operation under any conditions, should comprise both steady-state and dynamic models that can be used for power system simulation. Although the CIM is currently addressing the requirement of dynamic information exchange

The work of F. Gómez was supported in part by the European ITEA3 openCPS project.

The work of L. Vanfretti was supported in part by the Engineering Research Center Program of the National Science Foundation and the Department of Energy under Award EEC-1041877 and in part by the CURENT Industry Partnership Program.

through IEC-61970-302 and IEC 61970-457, this still might lead to the exchange of ambiguous models as it is explained in a previous work (Vanfretti, Li, Bogodorova, & Panciatici, 2013). The use of Modelica for dynamic model exchange may help in addressing the challenges as described next.

1.2 Previous Work

Specialized tools that allow the modeling and simulation of multi-domain systems for power system analysis have been created (Nicolet, Sapin, Simond, Prenat, & Avellan, 2001; Sapin, 1995), however they do not support power grid modeling for stability-analysis or the capability to simulate large grids.

The authors of (Vanfretti et al., 2013) and (Gómez, Vanfretti, & Olsen, 2015) have shown that Modelica language is able to cope with the exposed ambiguous model sharing issue while facilitating the access and/or modification of models at the "equation-level". Some additional advantages of Modelica are the open distribution of several libraries meant to represent physical systems, and the fact that models are independent from IDEs and solvers (Gómez et al., 2015). In addition, Modelica tools are now supporting the required numerical techniques to simulate large power grids (Braun, Casella, & Bachmann, 2017; Casella, Leva, & Bartolini, 2017; Dassault Systemes, 2018).

In addition, because turbine manufacturers already make extensive use of Modelica for thermo-mechanical and control modeling of gas turbines (Johansson, 2016), it becomes attractive to adopt a multi-domain modeling approach using Modelica in order to enhance dynamic characteristics of gas turbines and the power system..

1.3 Contributions

The work reported in this paper was carried out within the ITEA3 OpenCPS (Open Cyber-Physical System Model-Driven Certified Development) project. The project aims to develop modeling and simulation toolchains that can be applied to cyber-physical and multi-domain systems (ITEA3, 2017).

In the second use case of the work package D5.3B, the benchmark case corresponds to multi-domain models of improved gas turbines coupled to the power grid to meet European standardization requirements for grid connection.

This paper presents the development of a multi-domain gas turbine and power grid equation-based model, required to test the functionalities of the OpenCPS toolchains. In more detail, first a Modelica multi-domain model comprising the physical model of the gas turbine, the governor and a Single Machine Infinite Bus (SMIB) power network was generated. Then, an analysis of the multi-domain system that includes a comparison with the GGOV1-based equivalent system was performed.

The modeling approach shown through the example of the integration of a multi-domain model of a gas turbine with the electric grid can be adopted in future scenarios of multi-domain power plant-to-network integration where more complex models of geothermal, combined-cycle or wind power plants, among other resources are being used. Power systems analysts will benefit from the availability of more accurate models as high penetration of intermittent renewable resources continues to challenge traditional power system operations, making their study difficult with traditional power system tools.

The paper is organized as follows: Section 1 provides a motivation to the problem, along with the previous work and contributions from this work. Section 2 starts with a description of the Modelica libraries used to develop the models. Subsequently, it continues with the presentation of the turbo-machinery and power system domain models, as well as the multi-domain model that is obtained by combining the models of the two previous domains. Section 3 explains the studies and simulations that have been carried out on the power system-only model and the multi-domain model for comparison purposes. Finally, Sections 4 and 5 present the results obtained and provide the conclusions, respectively.

2 Equation-Based Models

2.1 Modeling Background

2.1.1 Package Structure

The equation based models were built or/and modified inside of a package structure in the Dymola F2016 Modelica IDE. The adopted package structure was conceived to classify the models in terms of the domain they belong to. The first two packages, namely *TurboMachineryDomain* and *PowerSystemDomain*, contain the physical gas turbine models and the electric power system models, respectively. A third package, called *MultiDomain*, comprises the results of merging components from the two former packages to obtain the multi-domain equation based models. In *PowerSystemDomain* only components from the *OpenIPSL* library are included. In addition to the SMIB network models, new stochastic variable load model and the gas turbine controls based on the *GGOV1* model are provided. On the other hand, the *TurboMachineryDomain* package was developed to comprise only elements from libraries specialized in gas turbines and other thermal power generation technologies such as *ThermoPower* (Casella & Leva, 2003), *ThermoSysPro* (El-Hefni, Bouskela, & Lebreton, 2011), *ThermoFluid* (Idebrant et al., 2003) or the *ThermalPower* library (Hübel et al., 2014). In this work only *ThermoPower* has been used.

2.1.2 OpenIPSL Library

OpenIPSL is an open-source Modelica library that can be used to create power system networks and then perform dynamic time-domain simulation. The Modelica language provides to this library the flexibility that is not common to find in other power system modeling and simulation tools (Baudette et al., 2018).

2.1.3 ThermoPower Library

ThermoPower is an open-source Modelica library developed at Politecnico di Milano. It provides components that can be used to model thermal power plants (Casella & Leva, 2003) (Casella, 2009).

The library has a package called Gas which contains the models of the gas turbine compressor, expansion turbine and combustion chamber. Their modeling description is based on DAE that are widely accepted in the turbine technology domain (Razak, 2007; Walsh & Fletcher, 2004).

More information about the library can be found in the official website (see URL: <https://casella.github.io/ThermoPower>).

2.2 Turbo-Machinery Domain Modelling

The *TurboMachineryDomain* package contains models which employ ThermoPower components. Its contents are organized in 3 sub-packages, namely *GTArrangements*, *GTModels* and *Tests*.

2.2.1 The GTArrangements package

As the name implies, the first package aimed to include the elementary gas turbine topologies. The *SingleShaftGT* model represents a single shaft gas turbine and it is based on the *Plant* model of the Brayton Cycle examples of *ThermoPower*. The Brayton Cycle is the thermodynamic cycle that describes the operation of a gas turbine. It is composed of at least four processes, three of which are associated with the components of a gas turbine, namely: compressor, combustion chamber and expansion turbine. The model only focuses on the internal components of the gas turbine. The parameters of the compressor, combustion chamber and turbine are propagated and therefore, the *SingleShaftGT* can be used as a generic block in the representation of gas power plants.

2.2.2 The GTModels package

The second package has the models that result from combining the basic parametrized gas turbine arrangement with given boundary conditions, sensors and actuators. The only example included to date is the complete ThermoPower Single Shaft Gas Turbine *ThPowerSSGT* model, which can be seen in Figure 1.

Due to unavailability of data, the design parameters and component characteristics of the *ThPowerSSGT* gas turbine model were not modified with respect to the ones of the original ThermoPower example. However, a

still simple but complete model of the fuel inlet valve that takes valve position as input instead of fuel mass flow reference was added. This change was needed to harmonize the physical model of the turbine with the simplified power system GGOV1-based turbine model.

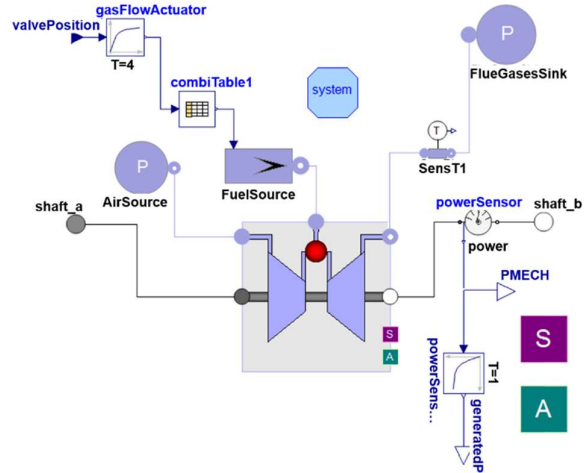


Figure 1. The Single Shaft Gas Turbine model built using ThermoPower components.

In order to build a valve model compatible to the simplified representations used in power system analysis, in particular the GGOV1 model, the fuel mass flow rate \dot{m}_{fuel} values required to obtain mechanical power from 0 to the maximum value of 10 MW were obtained through simulations and are shown in Table 1. In power system simulations, P_{mech} , is used as the output of the turbine. Hence, to relate the output mechanical power P_{mech} (in per unit) with the fuel inlet valve position $\theta_{fuel\ valve}$ (also in per unit), the following expression was used:

$$\theta_{fuel\ valve} = P_{mech}/K_{turb} + W_{fnl} \quad (1)$$

where K_{turb} is the gas turbine gain and W_{fnl} is the fuel mass flow rate at no load conditions (in per unit). Evaluating the parameters for a range of 0-10 MW gives a look-up table whose values are shown in Table 1.

Table 1. Fuel inlet valve model design data with $K_{turb} = 1.5$ and $W_{fnl} = 0.15$.

| P_{mech} (MW) | P_{mech} (pu) | $\theta_{fuel\ valve}$ (pu) | \dot{m}_{fuel} (kg/s) |
|-----------------|-----------------|-----------------------------|-------------------------|
| 0 | 0 | 0.150 | 1.845 |
| 1 | 0.1 | 0.217 | 1.919 |
| 2 | 0.2 | 0.283 | 1.989 |
| 3 | 0.3 | 0.350 | 2.059 |
| 4 | 0.4 | 0.417 | 2.129 |
| 5 | 0.5 | 0.483 | 2.199 |
| 6 | 0.6 | 0.550 | 2.270 |
| 7 | 0.7 | 0.617 | 2.341 |
| 8 | 0.8 | 0.683 | 2.413 |
| 9 | 0.9 | 0.750 | 2.485 |
| 10 | 1.0 | 0.817 | 2.558 |

Finally, the curve $\dot{m}_{fuel} = f(\theta_{fuel\ valve})$ from Table 1 was specified in the model as the look-up table that is shown in Figure 1 (see combiTable1D).

2.3 Power System Domain Modelling

This section provides an overview of the grid, load and control models which are based on OpenIPSL library components.

2.3.1 Generation Groups

The benchmark multi-domain model of use case 2 of the OpenCPS project work package (i.e. improved gas turbines coupled to the power grid) required different modeling and simulation scenarios for the SMIB network model. Two examples of these scenarios are a SMIB model without controls and a SMIB model with only excitation system. They were made available inside the sub-package *Generation_Groups* of the *PowerSystemDomain* package. This sub-package also includes a model of the “infinite bus” modeling construct which is typically used in power systems to represent a strong external system.

2.3.2 Controls

The GGOV1 is one of the so-called turbine-governor models that analysts use in power system dynamic studies. This is a generic model with blocks to represent thermal turbines that are controlled by a PID governor (proportional, integral, derivative). It also includes blocks that represent the dynamics of the fuel system, acceleration limiter, load limiter by exhaust temperature control, valve position and supervisory load controller.

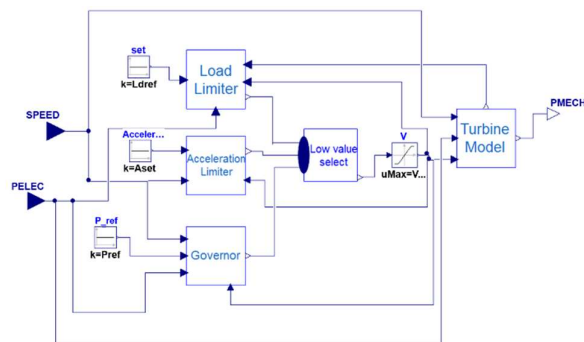


Figure 2. Modified GGOV1 Turbine Governor model.

The GGOV1 model implementation of the *OpenIPSL* library was refactored (i.e. modularized functions into internal blocks) so to fit the needs of the studies of this work.

As shown in Figure 2, refactoring was applied on the GGOV1 model to explicitly show its internal functionalities using internal blocks. This means that a separate model was created for each of the three controls logics that are inside of the GGOV1 model, namely the load limiter, the acceleration limiter and the main governor. Another model was developed to represent only the turbine, thus obtaining a convenient way to re-

use the models when a certain study requires to modify an internal block (e.g. only the turbine or the governor) instead of the entire model.

2.3.3 Network Models

A SMIB network model was developed for each of the generation groups described in Section 2.3.1. Figure 3 shows the SMIB network case where the generation group has no controls.

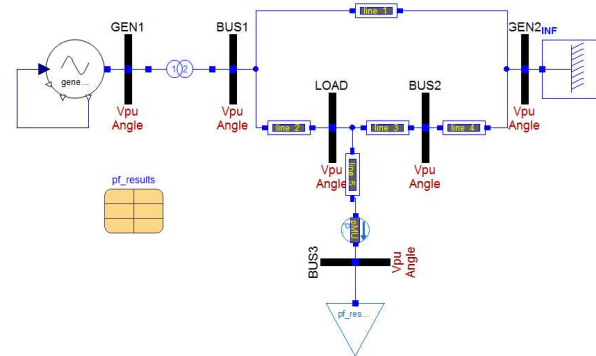


Figure 3. SMIB network model with no turbine governor model.

The initial voltage magnitude, voltage angle, active power and reactive power values of the generators, the load and the buses are specified by means of the record *pf_results*, which were obtained using an identical representation of the network using PSS/E, a domain specific tool (Siemens AG, 2018).

2.3.4 Variable Load Model

As it can be noticed from Figure 3, the SMIB network model also includes a variable load component. It can behave deterministically or stochastically, where the latter requires a stochastic signal as an input.

This model has now been included in the *OpenIPSL* library as *OpenIPSL.Electrical.PSSE.Load_ExtInput*. It is similar to the *.Load_variation* model, with the main difference being that it has a real input for modulation Error! Reference source not found.. Therefore, the new model has a component that allows for active power modulation in addition to the component that represents the physical load variability. The second component is adjusted by the parameters d_p (active load variation), d_t and $t1$ (start time and time duration of load variation), while the former relies on the noise injection source that is connected to this model (through input u). Noise injections have also been included in *OpenIPSL*, and are available under *OpenIPSL.Electrical.Loads.PSSE.NoiseInjections*. The simulation results where the load models exhibit a stochastic behavior driven at u are out of the scope of this paper, (see (Aguilera, Vanfretti, & Gómez, 2018)).

2.4 Multi-Domain Model

A SMIB network model and a governor block model from the *PowerSystemDomain* package was combined with the physical model of a gas turbine from the *TurboMachineryDomain* package. The result of this procedure gives the so-called multi-domain model that can be appraised in Figure 4.

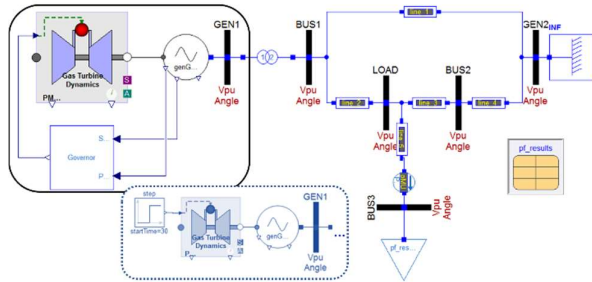


Figure 4. Multi-domain SMIB model.

New generation group sets have been created to allow the connection between the generator and the detailed gas turbine model. Even though these groups still rely on the previously defined groups of the *PowerSystemDomain* package (see Section 2.3.1), they also include an interface block. The function of this new block is to relate the rotational mechanics (flange internal variables) of the gas turbine model with the generator mechanical power and speed.

3 Simulation Studies

In this section, the simulations and studies that were applied on the equation-based models will be described. They include the identification of a GGOV1 turbine model, a frequency-domain analysis of the gas turbine models and simulations of the models when they are subject to a load change. An emphasis has been placed on the comparative analysis between the multi-domain model and the model that uses GGOV1 model that is a power system domain aggregate turbine-governor representation.

3.1 Study and Simulation Cases

This section begins with a description of the identification process of the GGOV1-based turbine model that fits the response of the *ThermoPower* detailed model. The results are necessary to carry out comparative studies on the resulting SMIB network models, namely a frequency domain analysis and the response to a load change event.

3.1.1 GGOV1-based Turbine Model Identification

The first step in the analysis to be done on the SMIB network models is the identification of the GGOV1 turbine model that is equivalent, in terms of its open-loop time response, to the *ThermoPower* model. An

open-loop test has been applied to the multi-domain SMIB model for that purpose.

The governor has been removed from the multi-domain SMIB model to apply a step change on the fuel mass flow rate in the gas turbine model. This can be observed in the box with dotted line of Figure 4, that replaces the box with solid line. Table 1 was employed to find the fuel mass flow rate values that give an output mechanical power change from 5 to 8 MW. Subsequently, a simulation was carried out in *Dymola* with a duration of 100 seconds, where the step change occurred after 30 seconds. The results were saved to proceed with the identification of the GGOV1-based turbine model.

The simulation output data has been imported in MATLAB as a .mat file. Then the system identification *ident* tool has been used to fit a GGOV1-turbine model that suits the reference model. The values of turbine gain K_{turb} and the no load fuel flow W_{fnl} were set to 1.5 and 0.15, respectively, as explained in Section 2.2.2. Additionally, the damping factor D_m was set to the typical value of 0. The decision to not consider this parameter is also based on the argument of (Pourbeik, 2013) that states: “A speed damping factor can be modeled to influence the temperature limit as a rather gross approximation of the speed dependence of the turbine rating. This is, however, not very accurate”. Thus, it has only been required to obtain the values of the parameters of the lead-lag transfer function T_b and T_c , together with the delay transport time T_{eng} .

From now on, the SMIB model that contains the physical model of the turbine will be referred to as **multi-domain model**. On the other hand, the SMIB network model using the GGOV1-based turbine model will be referred to as **power system-only model**.

3.1.2 Gas Turbine Models Frequency Domain Analysis

A first inspection of the differences between the expected response of the multi-domain model and the power system-only model can be carried out through pole/zero analysis. This study requires the linearization of the physical turbine model around a given operating point, which can be performed automatically from *Dymola* using the Modelica Linear Systems 2 library (Baur, Otter, & Thiele, 2009).

To understand the impact of the eigenvalues on the response of each model, the contribution of the poles in the states obtained after linearization has been identified. The results are discussed in Section 3.2.2.

3.1.3 Load Change Event Simulations

The next step is to verify the time-domain response of the models under a load change.

A simulation of 100 seconds was performed on both the multi-domain and power-system model (using the identified parameters as described in Section 3.1.1),

with the same governor model. The active power of the load was increased by 0.2 pu after 30 seconds of simulation, and was set back again to the original value after 20 seconds, in order to evaluate the turbine response to a sudden load change.

Successive simulations were carried out with the objective of evaluating the performance of the models at different operating points. Specifically, the load active power as well as the dispatched power from the generator were increased from 5 to 9.8 MW, in steps of 0.1 MW.

The load active power and dispatched generator power parameter sweep required the computation of 49 power flow solutions for the initialization of the network models. The solution sets were supplied in the form of records to conform with the description of Section 2.3.3. Python scripts were used to automatically generate the Modelica records. The scripts consist of a modified version of the toolset used to get the Nordic 44-bus system simulation results published in (Vanfretti, Rabuzin, Baudette, & Murad, 2016) and (Vanfretti et al., 2017).

In order to better quantify the time domain response differences of the two models, the settling times were computed. This was performed for the first simulation scenario, when the load model did not include the noise. Results are presented in Section 3.2.3 and discussed in Section 4.

3.2 Results

The results of the studies and simulations performed on the models are presented in this section.

3.2.1 Model Identification

A GGOV1-based turbine model with one pole and one zero with no time delay was identified. The resulting transfer function is:

$$g_4(s) = K_{turb} \frac{1 + T_c s}{1 + T_b s} = 1.5 \cdot \frac{1 + 0.115s}{1 + 0.141s} \quad (2)$$

The same step change on the fuel mass flow rate was applied on both the reference multi-domain model and the power system-only model without the governor. Figure 5 shows the output mechanical power plots from the turbine components of the models.

3.2.2 Eigenanalysis

The gas turbine models can be described in state space form as:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \quad (3)$$

In the case of the GGOV1-based turbine system of the Power System-only model, the system vectors and matrices are defined as:

$$u: valve_{position} \quad y: P_{mech} \quad (4)$$

$$x = \begin{pmatrix} g_4(s) \cdot x \\ gasFlowActuator.y \end{pmatrix}$$

$$\begin{aligned} A &= \begin{pmatrix} -7.092 & 11.123 \\ 0 & -0.25 \end{pmatrix} & B &= \begin{pmatrix} 0 \\ 0.25 \end{pmatrix}^{-1} \\ C &= \begin{pmatrix} 1.835 \times 10^6 \\ 12.807 \times 10^6 \end{pmatrix} & D &= 0 \end{aligned} \quad (5)$$

It can be easily found from the system state space equation that there are only real eigenvalues. They are shown on Table 2 together with their contribution on the identified systems states.

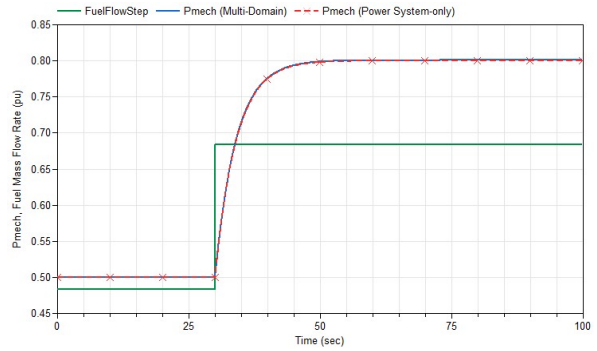


Figure 5. Open-Loop test to verify the response of the identified model, w.r.t. the multi-domain model.

Table 2. Real eigenvalues of gas turbine in the power system-only model.

| Eigenvalue | T(s) | Contribution to states | |
|----------------|-------|------------------------|------------------|
| | | State | Contribution (%) |
| $p_1 = -7.092$ | 0.141 | $g_4(s) \cdot x$ | 100 |
| $p_2 = -0.25$ | 4.0 | $g_4(s) \cdot x$ | 61.9 |
| | | $gasFlowActuator.y$ | 38.1 |

The system has only a zero $z_1 = -8.685$ with $T(s) = 0.115$.

Linearization was performed on the detailed gas turbine system of the multi-domain model at $t=0$. The state vector is as follows:

$$x = [CC.fluegas.p, \quad CC.fluegas.T, \quad CC.fluegas.X_1, \quad CC.fluegas.X_2, \quad CC.fluegas.X_3, \quad CC.fluegas.X_4, \quad CC.fluegas.X_5, \quad CC.T_m, \quad gasFlowActuator.y, \quad speedSource.\phi, \quad gasFlowActuator.y, \quad speedSource.\phi]^T \quad (6)$$

As can be seen in equation 6, most system states are associated with the combustion chamber of the turbine (CC). These states are the chamber wall temperature (T_m) and the pressure (p), temperature (T) and molar composition (X_1 - X_5) of the gases at the chamber outlet (i.e. flue gases).

System vectors B and C are now defined as follows:

$$B = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0.25 \quad 0]^T$$

$$C = [78.860, \quad 20707.3, \quad 3.939 \times 10^7, \\ 2.918 \times 10^7, \quad 7.291 \times 10^7, \\ 3.163 \times 10^7, \quad 4.456 \times 10^7, \\ 0, 0, 0] \quad (7)$$

Matrix A is given by:

$$A = \begin{bmatrix} -1.2 \times 10^{-3} & -3.98 \times 10^5 & -8.6 \times 10^8 & -6.9 \times 10^8 & -1.5 \times 10^9 & -6.3 \times 10^8 & -9.9 \times 10^8 & 0.31 & 2.6 \times 10^8 & 0 \\ -0.36 & -1.1 \times 10^4 & -3.2 \times 10^5 & -2.5 \times 10^5 & -5.6 \times 10^5 & -2.3 \times 10^5 & -3.6 \times 10^5 & 0.5 \times 10^{-3} & 3.9 \times 10^5 & 0 \\ -5.2 \times 10^{-6} & 1.5 \times 10^{-14} & -0.96 \times 10^5 & 2.2 \times 10^{-1} & 4.9 \times 10^{-11} & 1.99 \times 10^{-11} & 3.1 \times 10^{-11} & 0 & -39.10 & 0 \\ -6.3 \times 10^{-9} & 1.3 \times 10^{-13} & 6.1 \times 10^{-12} & -0.96 \times 10^5 & 1.1 \times 10^{-11} & 4.5 \times 10^{-12} & 7.03 \times 10^{-12} & 0 & -0.048 & 0 \\ 2.7 \times 10^{-6} & 4.9 \times 10^{-14} & -1.6 \times 10^{-11} & -1.3 \times 10^{-11} & -963.17 & -1.2 \times 10^{-11} & -1.9 \times 10^{-11} & 0 & 20.63 & 0 \\ 3.4 \times 10^{-6} & -2.6 \times 10^{-12} & -1.5 \times 10^{-12} & -1.2 \times 10^{-12} & -2.7 \times 10^{-12} & -963.17 & -1.7 \times 10^{-12} & 0 & 25.49 & 0 \\ 9.3 \times 10^{-7} & -3.6 \times 10^{-11} & -1.5 \times 10^{-11} & -1.2 \times 10^{-11} & -2.7 \times 10^{-11} & -1.1 \times 10^{-11} & -963.17 & 0 & -6.97 & 0 \\ 0 & 0.05 & 0 & 0 & 0 & 0 & 0 & -0.05 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The system has also only real eigenvalues as it is shown in Table 3.

Table 3. Real eigenvalues of gas turbine in the multi-domain model.

| Eigenvalue | T(s) | Relevant contribution to states | |
|---------------------------|----------------------|---------------------------------|------------------|
| | | State | Contribution (%) |
| $p_1 = -1.56 \times 10^3$ | 6×10^{-4} | CC.fluegas.p | 99.9 |
| $p_2 = -9.63 \times 10^2$ | 0.001 | CC.fluegas.T | 97.2 |
| $p_3 = -9.63 \times 10^2$ | 0.001 | CC.fluegas.T | 75 |
| | | CC.fluegas.X _g | 12.2 |
| $p_4 = -9.63 \times 10^2$ | 0.001 | CC.fluegas.T | 94.8 |
| | | CC.fluegas.X _g | 5.5 |
| $p_5 = -9.63 \times 10^2$ | 0.001 | CC.fluegas.T | 86.4 |
| | | CC.fluegas.X _g | 5.5 |
| $p_6 = -9.63 \times 10^2$ | 0.001 | CC.fluegas.T | 99.4 |
| $p_7 = -7.98 \times 10^2$ | 1.3×10^{-3} | CC.fluegas.p | 99.9 |
| $p_8 = -0.25$ | 4.000 | CC.fluegas.p | 99.6 |
| $p_9 = -0.05$ | 20.00 | CC.T _m | 100 |
| $p_{10} = 0$ | --- | speedSource.g | 100 |

Finally, the gas turbine of the multi-domain model has the zeroes shown in Table 4. The poles and zeros plots of the gas turbine systems from both models can be observed in Figures 6 and 7.

Table 4. Zeros of the gas turbine in the multi-domain model.

| Zero | Amount | T(s) |
|--------------------------------|--------|------------------------|
| $z_1 = -9.644 \times 10^2$ | 1 | 0.001 |
| $z_i = -9.632 \times 10^2$ | 4 | 0.001 |
| $z_6 = -6.441 \times 10^2$ | 1 | 1.6×10^{-3} |
| $z_7 = -0.050$ | 1 | 20.000 |
| $z_8 = -6.651 \times 10^{-14}$ | 1 | 1.503×10^{13} |

Table 5. Branch data of the SMIB network model ($V_b = 13.8$ kV).

| From bus | To bus | R (pu) | X (pu) |
|----------|--------|----------------------|--------------------|
| GEN1 | BUS1 | 0 | 0.150 |
| BUS1 | GEN2 | 1×10^{-5} | 0.200 |
| BUS1 | LOAD | 3×10^{-5} | 0.060 |
| LOAD | BUS2 | 3.5×10^{-4} | 0.070 |
| BUS2 | GEN2 | 3.5×10^{-4} | 0.070 |
| LOAD | BUS3 | 0 | 1×10^{-5} |

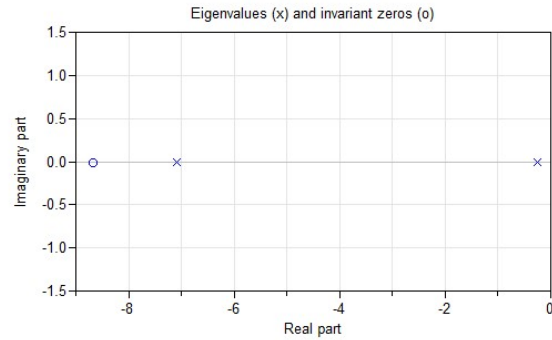


Figure 6. Poles and zeros of the gas turbine in the power system-only model.

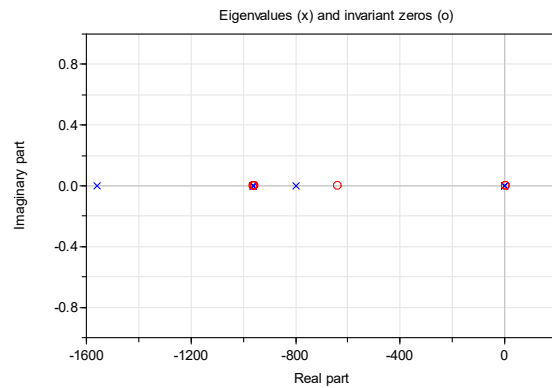


Figure 7. Poles and zeros of the gas turbine in the multi-domain model.

3.2.3 Time Response to Load Change

The governor was added to the multi-domain and power system-only models to evaluate their time response to a load change. The applied test was presented in Section 3.1.3 and the model parameters can be found in Tables 5-7. This section shows the results of the time response simulations. The simulations were performed with the variable step DASSL solver and a tolerance of 1×10^{-4} .

Table 6. Parameters of generated at BUS1 ($V_b = 13.8$ kV).

| Parameter | Gen-1 | Parameter | Gen-1 |
|----------------|--------|-----------------|-------|
| Capacity M_b | 10 MVA | X_q | 1.35 |
| T'_{d0} | 5.00 | X'_d | 0.30 |
| T''_{d0} | 0.05 | X''_q | 0.60 |
| T'_{q0} | 0.70 | $X''_d = X''_q$ | 0.20 |
| T''_{q0} | 0.10 | X_l | 0.12 |
| Inertia H | 4.00 | $S_{1,0}$ | 0.10 |
| Damping D | 0 | $S_{1,2}$ | 0.50 |
| X_d | 1.41 | R_d | 0 |

Figure 8 shows a plot of the mechanical power delivered by the gas turbine components. Special attention was also given to the response of the system frequency and the electrical power of the generator. The corresponding plots can be seen in Figures 9 and 10, respectively.

Then the simulation was repeated several times to measure the settling times, at different operating points of the generator. Figure 11 first shows a plot of the calculated settling times in both multi-domain and power system-only models as a function of the load/generator active power. The plot at the bottom of the figure shows the difference between the settling times obtained for the power system-only model and the settling times of the multi-domain model.

Table 7. GGOV1 governor-only parameters.

| Parameter | Value | Parameter | Value |
|-------------------|-------|--------------------|-------|
| R (pu) | 0.04 | K_{turb} (pu) | 1.50 |
| T_{pelec} (sec) | 1.00 | D_m (pu) | 0.00 |
| max_{err} (pu) | 0.05 | $K_{imw, db}$ (pu) | 0.00 |
| min_{err} (pu) | -0.05 | V_{max} (pu) | 1.00 |
| K_{pgov} (pu) | 10.00 | V_{min} (pu) | 0.10 |
| K_{igov} (pu) | 5.00 | W_{fml} (pu) | 0.15 |
| K_{dgo} (pu) | 0.00 | FLAG | 0 |
| T_{dgo} (sec) | 1.00 | | |

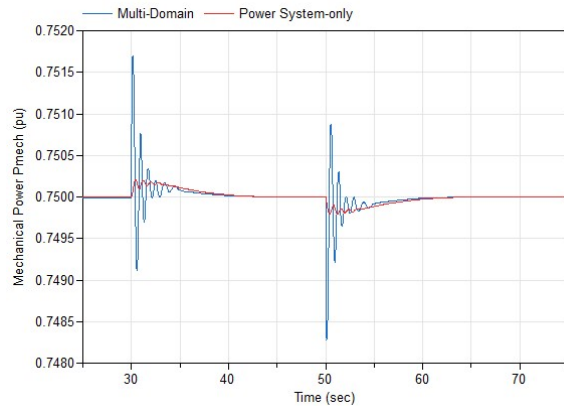


Figure 8. Mechanical power response comparison.

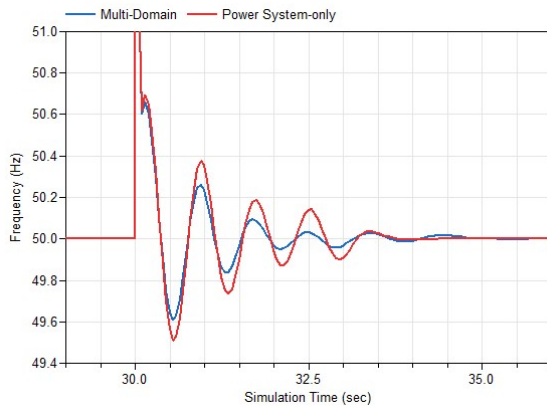


Figure 9. Frequency response comparison.

4 Discussion

The first evidence of the differences between the power system-only model and the multi-domain models can be found in the frequency analysis results from Section 3.2. The higher number of poles and states identified in the

linearized multi-domain model shows that the GGOV1 model used in the power system-only model will lead to loss of information about the physical dynamics of the turbine.

The information reported in Table 3 is particularly useful when it comes to giving a better physical explanation to the behavior of the model. First, the list of states allows to appreciate the relevance of the heating process in the gas turbine dynamics. Seven out of nine states were related to thermodynamic properties in the boundaries of the combustion chamber (denoted as CC in the state vector). However, six poles are cancelled with zeros, and thus only three poles deserve special attention.

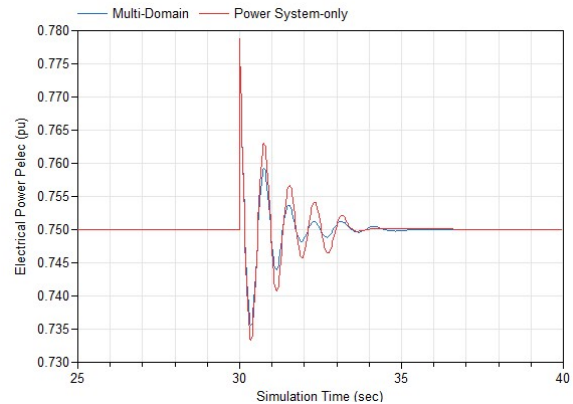


Figure 10. Electrical power response comparison.

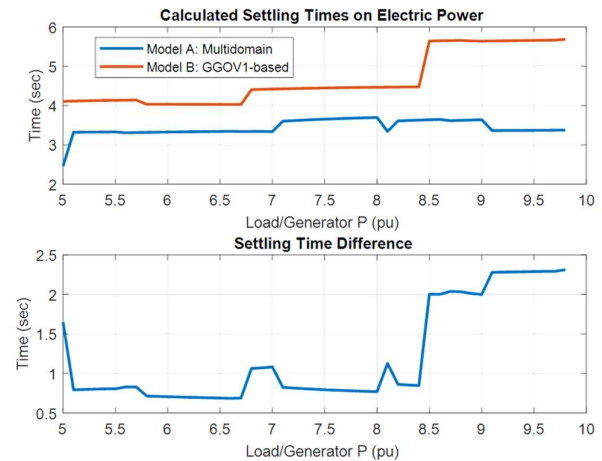


Figure 11. Calculated settling times in the electrical power response.

One of those poles is related to the fuel system actuator transfer function, which is also present in the simpler GGOV1-based model. As it can be seen in Table 3, the other two poles have a high contribution in the pressure of the flue gases at the exhaust of the combustion chamber. From theory, thermodynamic state of a gas is determined by two properties such as pressure and temperature, in addition to the molecular composition. The pressure of the flue gases (i.e. the

relevant state in the detailed gas turbine model) is directly linked to the Brayton Cycle and to the turbine's operation characteristics.

Figures 6 and 7 together with Tables 2 and 3 lead to another significant finding. In general, the explicit model from ThermoPower provides a higher bandwidth resolution in behavior modeling that is not possible with the GGOV1-based model. Therefore, this shows how a multi-domain model will be more suitable for transient stability studies (e.g. fault analysis, control design, etc.)

The effects on the time response of the models can be first examined in Figures 8 to 10. The load change event influences the system frequency, which is measured closed to the load bus, and is shown in Figure 9. Even if it is for a short time (around 2 sec), the frequency experiences a maximum deviation of up to 1 Hz. Such frequency excursions are unacceptable in practice as protective over/under frequency protection systems can be triggered. Observe that the power system-only model results give an over-estimation of the expected frequency, and thus, any control/protection system design using such model may give unexpected results in practice. In Figure 9, the frequency of the power system-only model goes beyond 49.6 Hz which is typically the limit for under-frequency protections, while the multi-domain model is below it, making the latter more suitable for model-based design.

The GGOV1 turbine model is not dependent on the shaft speed and therefore, the changes on the mechanical power of Figure 8 are due to the governor's response. However, this is not in the case of the multi-domain turbine model. That explains why the model produces an additional oscillatory behavior on the mechanical power that cannot be observed in the GGOV1-based turbine model response. Also, note that the output mechanical power is grossly under-estimated by the power system-only model w.r.t the multi-domain model.

The electrical power can be used to examine the impact of the gas turbine model response on the generator's electric power output (see Figure 10). Nevertheless, it is important to keep in mind that the electrical power is also directly influenced by the speed. The settling times of this variable were calculated for different values of the load/generator power and then plotted in Figure 11. The results show a higher amplitude in the frequency response when the GGOV1-based turbine model is employed. Although the settling times difference between the two models' response keep fairly constant, an increase is obtained for active power values greater or equal than 0.85 pu. It has been found that the cause of this performance is the saturation of the fuel actuator limiter in the GGOV1-based turbine model.

5 Conclusions

The following conclusions and recommendations can be drawn from this work:

- A multi-domain model has been derived to allow simulations of detailed representations of gas turbines and the electric power grid. Although the models are simple (due to the lack of available modeling information) the methodology provides a framework for future studies with multi-domain models in power systems.
- Differences in the simple turbine model (GGOV1) and the multi-domain explicit turbine model have been shown. A relevant source of that difference is the *representation of the speed influence on the gas turbine dynamics*. The study was, however, limited by the lack of measurements that could have served as a reference for the model's tuning and validation. It would also be of value to analyze the differences between the models in other power network variables and not only in the generator response.

This work gives a proof-of-concept on the use of Modelica for joint modeling of complex energy sources without the loss of information that traditional power system approaches incur in. The multi-domain approach is thus valuable for power system analysts, especially those dealing with controller design and dynamic performance analysis.

Acknowledgement

The authors would like to thank Francesco Casella for his assistance with questions regarding ThermoPower. In the same way, they would like to acknowledge the contributions of Tin Rabuzin in initial stage of this work.

References

- Aguilera, M., Vanfretti, L., & Gómez, F. (2018). Experiences in power system multi-domain modeling and simulation with modelica & FMI: The case of gas power turbines and power systems. In *2018 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)* (pp. 1–6). IEEE. <https://doi.org/10.1109/MSCPES.2018.8405397>
- Baudette, M., Castro, M., Rabuzin, T., Lavenius, J., Bogodorova, T., & Vanfretti, L. (2018). Open-Instance Power System Library Update 1.5 to "iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations. *SoftwareX*. <https://doi.org/10.1016/j.softx.2018.01.002>
- Baur, M., Otter, M., & Thiele, B. (2009). Modelica Libraries for Linear Control Systems LinearSystems library. *Proceedings of the 7th International Modelica Conference*, 20–22.
- Braun, W., Casella, F., & Bachmann, B. (2017). Solving large-scale Modelica models: new approaches and experimental results using OpenModelica. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017* (pp. 557–563).

- Carnegie Mellon University. (2013). *Managing Variable Energy Resources to Increase Renewable Electricity's Contribution to the Grid*.
- Casella, F. (2009). Object-oriented modelling of power plants: a structured approach. *IFAC Proceedings Volumes*, 42(9), 249–254.
- Casella, F., & Leva, A. (2003). Modelica open library for power plant simulation: design and experimental validation. In *Proceeding of the 2003 Modelica conference, Linkoping, Sweden*.
- Casella, F., Leva, A., & Bartolini, A. (2017). Simulation of Large Grids in OpenModelica: reflections and perspectives, 227–233.
- CEN-CENELEC-ETSI Smart Grid Coordination Group. (2012). Smart Grid Reference Architecture, (November), 1–46. Retrieved from ftp://ftp.cenelec.eu/EN/EuropeanStandardization/HotTopics/SmartGrids/Reference_Architecture_final.pdf
- Dassault Systemes. (2018). Dymola Sparse Solvers for Large-Scale Simulations.
- De Mello, F. P., & Ahner, D. J. (1994). Dynamic models for combined cycle plants in power system studies. *IEEE Transactions on Power Systems*, 9(3).
- El-Hefni, B., Bouskela, D., & Lebreton, G. (2011). Dynamic Modelling of a Combined Cycle Power Plant with ThermoSysPro. *Proceedings of the 9th Modelica Conference*, 365–375.
- Gómez, F. J., Vanfretti, L., & Olsen, S. H. (2015). Binding cim and modelica for consistent power system dynamic model exchange and simulation. In *Power & Energy Society General Meeting, 2015 IEEE* (pp. 1–5).
- Gomez, F., Vanfretti, L., & Olsen, S. H. (2018). CIM-Compliant Power System Dynamic Model-to-Model Transformation and Modelica Simulation. *IEEE Transactions on Industrial Informatics*, 3203(c), 1–1. <https://doi.org/10.1109/TII.2017.2785439>
- Hannett, L. N., & Khan, A. H. (1993). Combustion turbine dynamic model validation from tests. *IEEE Transactions on Power Systems*, 8(1), 152–158.
- Hübel, M., Berndt, A., Meinke, S., Richter, M., Mutschler, P., Hassel, E., ... Funkquist, J. (2014). Modelling a lignite power plant in modelica to evaluate the effects of dynamic operation and offering grid services. In *Proceedings of the 10th International Modelica Conference; March 10-12; 2014; Lund; Sweden* (pp. 1037–1046).
- Idebrant, A., Näs, L., Ab, M. E., Industrial, A., Ab, T., Bachmann, B., ... Fritzson, P. (2003). Gas Turbine Applications using ThermoFluid. *Proceedings of the 3rd International Modelica Conference*.
- IEA. (2016). *Energy, Climate Change & Environment - 2016 Insights*.
- ITEA3. (2017). OpenCPS - Open Cyber-Physical System Model-Drive Certified Development. Retrieved June 19, 2017, from <https://itea3.org/project/opencps.html>
- Johansson, T. (2016). Simulation of gas channel temperatures during transients for SGT-800.
- Nicolet, C., Sapin, A., Simond, J. J., Prenat, J. E., & Avellan, F. (2001). A new tool for the simulation of dynamic behaviour of hydroelectric power plants. In *Proceedings of the 10th International Meeting of WGI, IAHR, Trondheim, Norway*.
- Pereira, L., Undrill, J., Kosterev, D., Davies, D., & Patterson, S. (2003). A new thermal governor modeling approach in the WECC. *IEEE Transactions on Power Systems*, 18(2), 819–829.
- Pourbeik, P. (2013). Dynamic models for turbine-governors in power system studies. *IEEE Task Force on Turbine-Governor Modeling*.
- Razak, A. M. Y. (2007). *Industrial gas turbines: performance and operability*. Elsevier.
- Rowen, W. I. (1983). Simplified mathematical representations of heavy-duty gas turbines. *Journal of Engineering for Power*, 105(4), 865–869.
- Rowen, W. I. (1992). Simplified mathematical representations of single shaft gas turbines in mechanical drive service. In *ASME 1992 International Gas Turbine and Aeroengine Congress and Exposition* (p. V005T15A001--V005T15A001).
- Sapin, A. (1995). Logiciel modulaire pour la simulation et l'étude des systèmes d'entraînement et des réseaux électriques.
- Siemens AG. (2018). PSS®E – high-performance transmission planning and analysis software. Retrieved from <https://www.siemens.com/1>
- Vanfretti, L., Li, W., Bogodorova, T., & Panciatici, P. (2013). Unambiguous power system dynamic modeling and simulation using modelica tools. In *Power and Energy Society General Meeting (PES), 2013 IEEE* (pp. 1–5).
- Vanfretti, L., Olsen, S. H., Arava, V. S. N., Laera, G., Bidadfar, A., Rabuzin, T., ... Gómez-López, F. J. (2017). An open data repository and a data processing software toolset of an equivalent Nordic grid model matched to historical electricity market data. *Data in Brief*, 11, 349–357. <https://doi.org/10.1016/j.dib.2017.02.021>
- Vanfretti, L., Rabuzin, T., Baudette, M., & Murad, M. (2016). iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations. *SoftwareX*, 5, 84–88.
- Walsh, P. P., & Fletcher, P. (2004). *Gas turbine performance*. John Wiley & Sons.
- Yee, S. K., Milanovic, J. V., & Hughes, F. M. (2008). Overview and comparative analysis of gas turbine models for system stability studies. *IEEE Transactions on Power Systems*, 23(1), 108–118.

Analysing the stability of an islanded hydro-electric power system

Dietmar Winkler

University of South-Eastern Norway, Norway, dietmar.winkler@usn.no

Abstract

Power system simulation is a large arena especially in connection with the large European power system. The challenges of large interconnected electrical power systems call for a sophisticated system modelling solution that can give comparable results. This led to project “iTesla – Innovative Tools for Electrical System Security within Large Areas” (iTesla 2016) which was funded by the European Commission. One result of that project was the open-source modelling library called “iTesla Power System Library - iPSL” (Vanfretti et al. 2016) which then later was forked and called “Open Instance Power System Library - OpenIPSL” (ALSETLab 2018). Those libraries are based on the open-source modelling language “Modelica” (Modelica Association 2017).

This paper presents the results of a Master’s thesis where Modelica was used in combination of the “OpenIPSL” library to model a small local distribution grid that is islanded.

It describes how to build the power system model using Modelica of a grid that is located in the Westfjord area of Iceland. That area of Iceland is only connected to the national grid by one transmission line. The reliability of the power supply is poor due to harsh weather conditions during winter.

Two models of the transmission system of the Westfjords were built. One is a base model with three generating units and one is an extended model with four generating units. Two different load scenarios were simulated. The result of which could give indicators as to what actions would help to keep the islanded grid stable.

Keywords: hydroelectric systems, electric power systems, modelling, modelica, open-source

1 Introduction

Iceland’s electrical energy sector has a strong focus on renewable energy and nearly all electrical energy produced is from renewable resources. Hydro power accounts for 72 % of the production. Iceland is the largest electrical power producer per capita in the world.

Being connected to the Icelandic power grid with only one transmission line makes the Westfjords dependent on the internal production of the region in cases where the connection to the national grid is lost. To improve the conditions the power production inside the area needs to be increased. The largest power station in the area is Mjólká. This power station consists of 3 generating units with a ca-



Figure 1. Overview of the transmission system with voltage levels: 132kV 66kV 33kV 11kV

capacity of approximately 13.2 MVA. The area is dependent on hydro power as renewable energy source for production of electrical energy as the area has little natural hot water resources for geothermal energy production.

The parts of the Westfjord transmission system including generating units, transmission lines, transformers, busses and loads were modelled using “OpenIPSL” (Open-Instance Power System Library) (ALSETLab 2018). The transmission system is simulated when the connection to the national grid is lost. As the frequency drops the individual loads are partly disconnected. It is of interest to find the disconnection sequence that gives the fastest stabilisation of frequency and voltage and what effect an additional production of a fourth generating unit, Hvesta, has on the stabilisation of the frequency and voltage.

The quality of the electricity, should be according to Regulation No. 1048/2004 on the quality of electricity and security of supply. This regulation states that the frequency shall be within 47 – 52 Hz all the time, and within 49.5 – 50.5 Hz 95 % of the time. An internal goal of the transmission line operator Landsnet is that the frequency is within 49.8 – 50.2 Hz 95 % of the time. The measurement used for assessing the frequency is the average frequency over a 10 s period. The regulation states the supplied voltage shall be within $\pm 10\%$ of the rated bus voltage. An exception is when supplied to power-intensive industries, where the limits are $+5/-9\%$. This is only applicable when assessing voltage quality of 220 kV lines, (Landsnet 2015).

2 System description

2.1 Transmission system

Figure 1 shows a part of the transmission system of the Westfjords in the northwestern part of Iceland.

This part of the transmission system consists of:



Figure 2. Power house of Mjólká 1 and Mjólká 2 (Westfjord Power Company 2018)

- Six transmission lines with different voltage levels.
- Four generating units, three at Mjólká (3.4 MVA, 8.5 MVA, 1.35 MVA) and one in Hvesta (1.7 MVA).
- Three loads located in Talknafjordur, Patreksfjordur and Bildudalur.

The Westfjords are connected to the national grid via the Geiradalur substation. This connection is essential for the area as the internal production of all of the Westfjords only covers about 60 % of the consumed power. The remaining 40 % of the consumed power is imported to the area by the national transmission system from the Geiradalur substation which in turn is connected to the national high-voltage ring line.

2.2 Mjólká power station

The Mjólká power station, located in Arnafjordur and is the largest power station in the Westfjords, with an average yearly production of 54 GWh. The Westfjords Power Company owns and operates the power station, which consist of three separate generating units. The first unit Mjólká 1 was built in 1956 and put into operation in 1958. Mjólká 2 was put into operation in 1975. Mjólká 1 and Mjólká 2 have a common power house, which can be seen in Figure 2.

Until 1980, when the Westfjords were connected to the national grid, Mjólká was the main power supply of the area. Mjólká 3 was constructed in 2010, and is located upstream of Mjólká 1. The recent years Mjólká 1 and Mjólká 2 have undergone turbine and generator upgrades. This led to an increased capacity of the power station by a total of 2.1 MW. All units have separate pressure shaft and utilise water from three different reservoirs. Mjólká 3 uses the reservoir of Mjólká 1 as tail water.

3 Modelling

3.1 Modelica

Modelica is an open-source high-level object-oriented, equation based modelling language for modelling of physical systems developed by the non-profit Modelica Association. The background for the development was the need for a standardised modelling language for reusable and exchangeable models. The Modelica Association has

also developed a standard library which consists of more than 1600 model components within several domains. In order to utilise the language, a modelling and simulation environment is needed. There exists both open-source and commercial tools like OpenModelica (OSMC 2018) and Dymola (Dassault Systèmes 2018), respectively.

3.2 OpenIPSL

The OpenIPSL (ALSETLab 2018) is an open-source library for modelling of electrical power systems. It is developed as an continuation of the iTesla project and is maintained by the ALSETLab research group. The library was used to model the following components:

- Generator
- Transformer
- Bus
- Power line
- Automatic Voltage Regulator (AVR)
- Power System Stabiliser (PSS)
- Turbine governor

The library also includes a tutorial and several application examples in addition to single components. The models are built in a drag and drop manner, setting up the models is fast and intuitive. The components are based on and validated against models from the existing power system software such as “Power System Simulator for Engineering (PSS/E)” (Siemens 2018) and “Power System Analysis Toolbox (PSAT)” (Milano 2018).

3.3 Model of transmission system

The model of the transmission system from Mjólká to Keldeyri consists of the four generating units Mjólká 1, 2, 3 and Hvesta, the transmission lines, the transformers and three loads. At Keldeyri the power is distributed to the loads which are located in Talknafjordur, Bildudalur and Patreksfjordur. The generating units consist of a PSS, an AVR, a turbine governor and a generator model. The Icelandic national grid is modelled as an infinite bus which is connected to the transmission line at Geiradalur. The infinite bus is a source which provides a constant voltage at a constant frequency and can provide and consume infinite amounts of active and reactive power. The model of the transmission system is made reusable so that it can be used as basis for future simulations of similar systems. All important parameters can be changed through a parameter record.

Two models of the transmission system were made, one model with the three Mjólká generating units and one extended model with an additional production from the Hvesta power station. Both are shown in Figure 3 and Figure 4, respectively.

The system consists of:

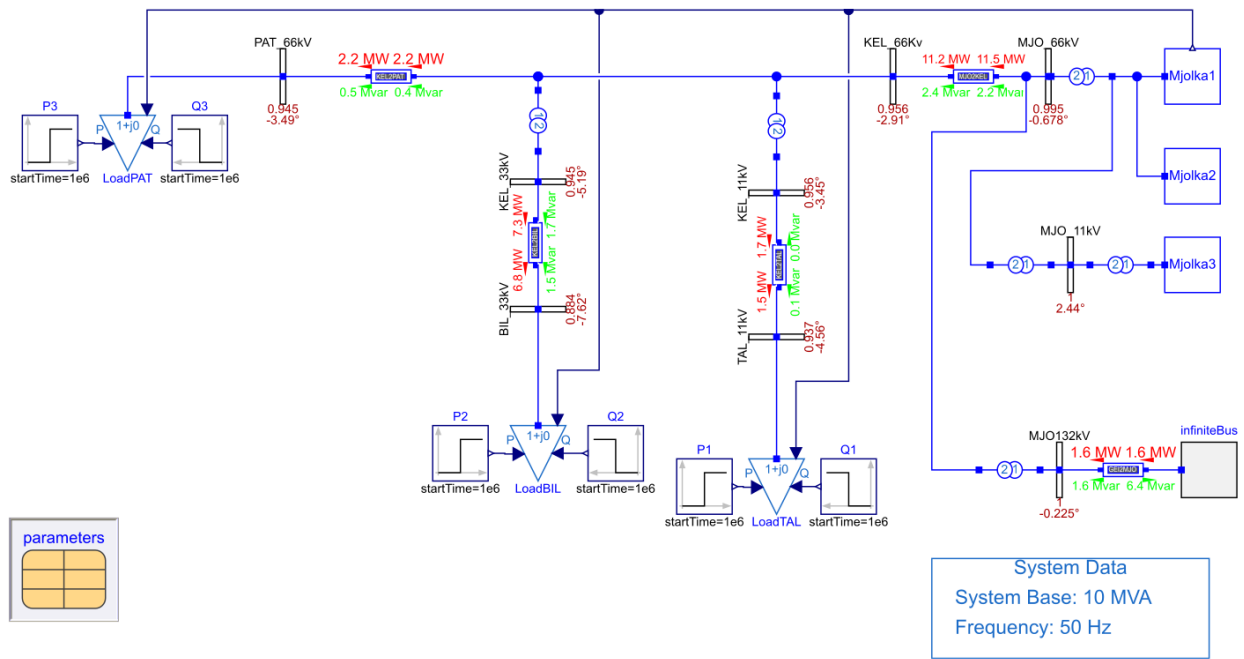


Figure 3. Three-generator model

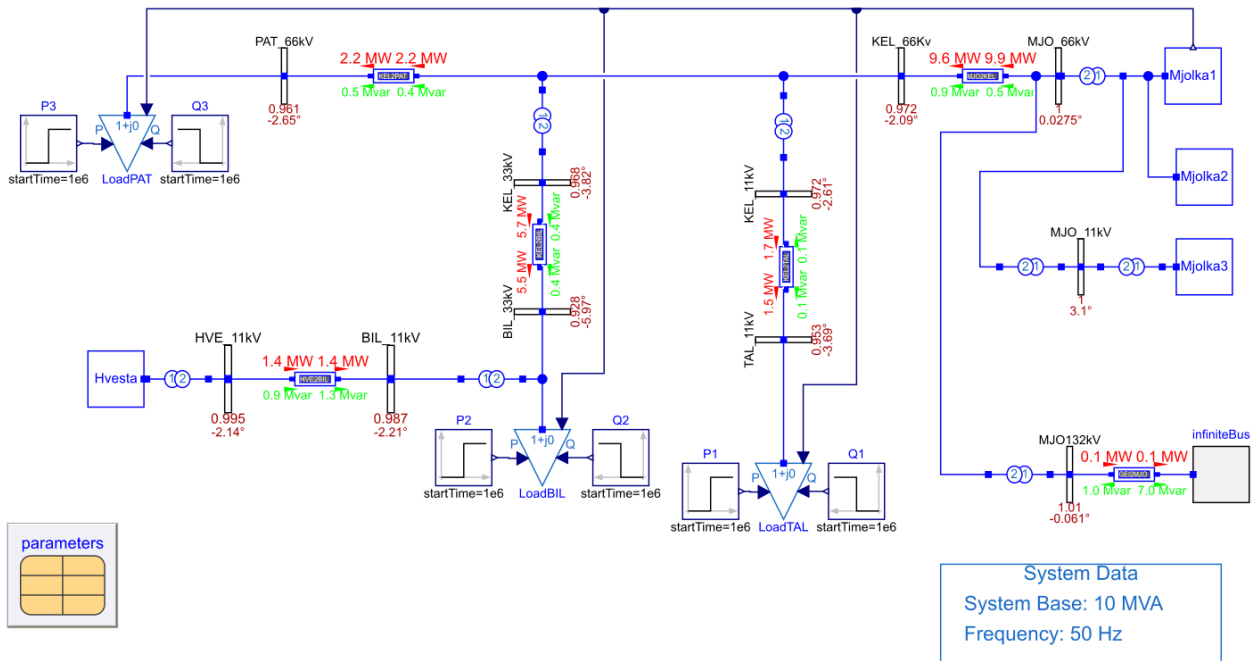


Figure 4. Four-generator model

- The transmission line from Geiradalur to Mjólká is a 132kV line which is stepped down to 66kV at the Mjólká substation and connected to the “MJO_66kV” bus where Mjólká 1, 2 and 3 are connected via a 6.3kV to 66kV transformer.
- Mjólká 3 generates at 0.4kV which is stepped up to 11kV and is then connected to the “MJO_11kV” bus which are transformed down to 6.3kV and is connected with Mjólká 1 and 2.

- The Mjólká substation and Keldeyri are connected by a 66kV line. At Keldeyri the power is distributed to the load centres at Bildudalur, Talknafjordur and Patreksfjordur. Talknafjordur is supplied by a 11kV line, Patreksfjordur is connected through a 66kV line and Bildudalur is supplied by a 33kV line.

In the extended model Hvesta power station is connected to Bildudalur via a 11kV line.

The parameters used in the model stem from a PSS/E

model of the Icelandic transmission system and were provided by the power company. These were used as basis for parameterisation of the models.

3.3.1 Generating units

The model used for the generating units consists of:

- PSAT 2nd order generator
- PSAT Turbine governor type 2
- PSAT AVR type 3
- PSAT PSS type 2

The structure of the model is shown in Figure 5.

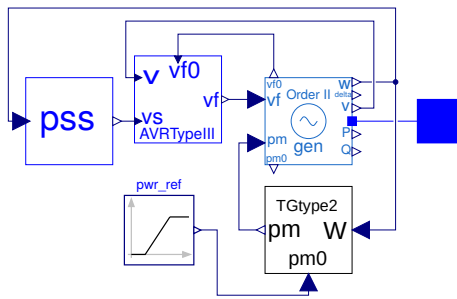


Figure 5. Generating unit model

3.3.2 Transmission lines

Five transmission lines are modelled in the base model and six lines in the extended model. Details and line names used in the models are given in Table 1.

Table 1. Overview of transmission lines in model

| Model name | From | To | Voltage [kV] | Length [km] |
|------------|------------|----------------|--------------|-------------|
| GEI2MJO | Geiradalur | Mjólká | 132 | 81 |
| MJO2KEL | Mjólká | Keldeyri | 66 | 50 |
| KEL2TAL | Keldeyri | Talknafjordur | 11 | 9 |
| KEL2BIL | Keldeyri | Bildudalur | 33 | 13 |
| KEL2PAT | Keldeyri | Patreksfjordur | 66 | 10 |
| HVE2BIL | Hvesta | Bildudalur | 11 | 4 |

The parameters from the PSS/E model used a different per-unit base of $S_{B_{PSS/E}} = 100 MVA$. The models in OpenIPSL on the other hand used $S_{B_{OpenIPSL}} = 10 MVA$. So all impedances needed to be corrected using (1).

$$z_{OpenIPSL} = z_{PSS/E} \frac{S_{B_{OpenIPSL}}}{S_{B_{PSS/E}}} \quad (1)$$

The line parameters for HVE2BIL and KEL2TAL were estimated and were based on parameters for KEL2BIL (taken from the PSS/E reference). The reactance for KEL2BIL was calculated in $\frac{\Omega}{km \cdot MW}$ in order to serve for a better estimation base for the calculation of HVE2BIL and KEL2TAL.

3.3.3 Transformers

The PSAT two-winding transformer models were used for all transformers.

3.3.4 Loads

The loads used a modified version of the PSAT LOADPQ model. This is a constant load where the load amount is entered as active load in MW and reactive load in $Mvar$.

The modifications done were:

- The system frequency and size of active and reactive loads are provided as inputs through connectors.
- A specified amount of the loads is automatically disconnected when system frequency drops below a user specified limit. Disconnected loads remain disconnected until the system frequency rises above a specified limit.

4 Simulation of the islanded grid

This section contains the results of simulations when the Westfjord area loses the connection to the national grid. As this happens the simulated grid operates as an islanded grid where all power consumed must be produced within the grid. It is known that the consumption of the area is larger than the production and therefore the frequency will drop after the disconnection. The voltage levels throughout the system will also be affected by less available reactive power. As the frequency drops, parts of loads in the area will automatically be disconnected at specified frequencies to maintain a balance in the consumption and production. It is of interest to see what sequence in which the loads are disconnected will give the fastest stabilisation of voltage and frequency. It is also of interest to see if the voltages and frequency are within the quality limits presented in the introduction.

Two main load scenarios have been simulated, one scenario with active and reactive loads, and one scenario with only active loads. Each scenario is simulated using both the model with three generation units and four generation units. This is done to examine what effect the additional production from Hvesta has on the stability of the system frequency and voltages compared to the model with only the Mjólká generating units. The two main scenarios are divided into sub-scenarios where different loads are disconnected to see which load gives the fastest stabilisation of system frequency and voltages. The voltages are measured at seven busses for the three generator model and nine busses for the four generator model.

All plots can be found in the appendix.

4.1 Scenario 1: Active and reactive loads

The initial loads used for simulation of Scenario 1 is given in Table 2.

Table 2. Initial loads for Scenario 1

| Load name | Active load [MW] | Reactive load [Mvar] |
|-------------|------------------|----------------------|
| LoadTAL | 2.5 | 0.1 |
| LoadBIL | 6.85 | 1.5 |
| LoadPAT | 2.16 | 0.54 |
| Total loads | 11.51 | 2.14 |

After the disconnection from the national grid the loads will be partially disconnected if the frequency drops below a specified limit. The disconnected loads remain disconnected throughout the simulation. For all simulations of Scenario 1 the connection to the national grid is lost at $t = 20s$. All generating units are running at maximum production from the start of the simulations.

4.1.1 Scenario 1.1: Disconnection of LoadPAT and LoadTAL

Figure 6 shows the resulting voltage and frequency plots for Scenario 1.1 simulated with the three generator model. The frequency starts to drop at 20 s when the connection to the national grid is lost. At 48 Hz, 50% of the active and reactive load of LoadPAT is disconnected and at 47 Hz, 50% of the active and reactive load of LoadTAL. The frequency and voltages stabilises after 105 s, this gives a stabilisation time, measured from time of disconnection, of 85 s. All voltages and the frequency stabilises within the quality limits except the voltage at BIL_33kV bus which stabilises at 0.88 pu. From 20 s to 26 s all voltage levels except MJO_66kV are below the quality limit.

4.1.2 Scenario 1.2: Disconnection of LoadBIL

Figure 7 shows the resulting voltage and frequency plots for Scenario 1.2 simulated with the three-generator model. The frequency drops down to 48 Hz, where 50% of the active and reactive load of LoadBIL is disconnected. The frequency and voltages stabilise after 55 s, this gives a stabilisation time from the disconnection of 35 s. All voltages and frequency stabilise within the quality limits. From 20 s to 31 s the voltages at all busses except MJO_66kV are below the quality limits.

4.1.3 Scenario 1 for four-generator model

For the four generator model, the loads of load Scenario 1 are not large enough to overload the transmission system. From Figure 8 it can be seen that the voltages and frequency are stabilised 8 s after the disconnection from the national grid. All voltages and frequency are within the quality limits for the voltage and the frequency except the very instance when the connection is lost.

4.2 Scenario 2: Active loads

The initial loads used for Scenario 2 are given in Table 3. In this scenario, the reactive parts of the loads have been neglected and the active power has been increased compared to Scenario 1. The power lines and transformers will still consume reactive power in this scenario.

Table 3. Initial loads for Scenario 2

| Load name | Active load [MW] |
|-------------|------------------|
| LoadTAL | 3.3 |
| LoadBIL | 6.85 |
| LoadPAT | 2.16 |
| Total loads | 12.28 |

After the disconnection from the national grid, the loads will be partially disconnected if the frequency drops below a specified limit. The disconnected loads remain disconnected throughout the simulation. For all simulations of Scenario 2 the connection to the national grid is lost at $t = 20s$. All generating units are running at maximum production from the start of the simulations.

4.2.1 Scenario 2.1: Disconnection of LoadPAT and LoadTAL

Figure 9 shows the resulting voltage and frequency plots for Scenario 2.1 simulated with the three-generator model. At 48 Hz, 50% of the active load of LoadPAT is disconnected and at 47 Hz, 50% of the active load of LoadTAL. The frequency and voltages stabilises after 75 s, this gives a stabilisation time from the disconnection of 55 s. All voltages and frequency stabilises within the quality limits. From 20 s to 34 s the voltages of TAL_11kV and BIL_33kV busses are below the voltage limits.

Figure 10 shows the resulting voltage and frequency plots for Scenario 2.1 simulated with the four-generator model. At 48 Hz, 50% of the active load of LoadPAT is disconnected.

For this simulation LoadTAL does not need to be disconnected to stabilise the frequency and voltages. The frequency and voltages stabilises after 90 s, which gives a stabilisation time from the disconnection of 70 s. All voltages and the frequency are within the quality limits, the voltage levels are higher than for the three-generator model

4.2.2 Scenario 2.2: Disconnection of LoadBIL

Figure 11 shows the resulting voltage and frequency plots for Scenario 2.2 simulated with the three-generator model. The frequency drops down to 48 Hz, where 50% of the active load of LoadBIL is disconnected.

The frequency and voltages stabilises after 50 s. This gives a stabilisation time from the disconnection of 30 s. All voltages and the frequency stabilise within the quality limits. All voltage levels except MJO_66kV are below limits in the time between 20 s to 27 s.

Figure 12 shows the resulting voltage and frequency plots for Scenario 2.2 simulated with the four-generator model. The frequency drops until 48 Hz , where 50% of the active load of LoadBIL is disconnected. The frequency and voltages stabilises after 70 s . This gives a stabilisation time from the disconnection of 50 s . All voltages and frequency stabilise within the quality limits. The voltages at MJO_66kV and HVE_11kV bus are above limits in the time between 47 s to 51 s and TAL_11kV is below the limits at 54 s .

5 Discussion

5.1 Summary Scenario 1

The additional production of Hvesta, manages to keep the system stable after the disconnection. For the three generator model the disconnection of LoadBIL gives the best results. The system stabilises faster, and all voltages stabilises within $\pm 10\%$ of the rated voltage. At steady state before the disconnection it can be seen from both simulations of the three generator model that the voltage at BIL_33kV is too low, this can be regulated locally for example by a tap changing transformer.

5.2 Summary Scenario 2

The additional production of Hvesta, causes the four-generator model to stabilise slower than the three generator model, as the frequency will drop slower. The four-generator model gives more stable voltage at the busses compared to the three generator model. The disconnection of LoadBIL gives the fastest stabilisation for both models.

5.3 Modelling challenges

Due to difficulties simulating the transmission system after losing the connection to the main grid using the load parameters as used in the PSS/E model, the size of the reactive load had been reduced in the simulations. For the same numerical reason the transformers reactance is assumed to be less than what is used in PSS/E. This can lead to inaccuracies in the simulation results of voltage levels at the busses compared to reality, where the voltage levels probably will be somewhat lower than what is shown in the simulations of Scenario 1. Still the simulations will give a good indication of the time it will take for the voltage levels to stabilise.

For the active power and the frequency this will give more accurate results, as the parameters used are equal to the PSS/E model for the active loads. The assumed turbine governor parameters are not good enough in cases where a large amount of the load is removed. The governor struggles with decreasing the production enough to allow the frequency to stabilise at 50 Hz . It can be assumed that with correctly tuned controls the frequency would settle at 50 Hz not at 50.2 Hz .

Acknowledgement

This paper is based on the Master's thesis by Kim Aars with the title "Simulation of load and fault scenarios in a hydro power system with island grid" (Aars 2017) finished in May 2017 at the University College of Southeast Norway.

This project was carried out in collaboration with Verkís Consultant Engineers, Iceland.

References

- Aars, Kim (May 31, 2017). "Simulation of load and fault scenarios in a hydro power system with island grid". Master's thesis. University College of Southeast Norway.
- ALSETLab (2018). *OpenIPSL*. URL: <http://openips1.org>.
- Dassault Systèmes (2018). *Dymola*. Modelon. URL: <http://www.dymola.com> (visited on 05/28/2016).
- iTesla (2016). *iTesla - Innovative Tools for Electrical System Security within Large Areas*. URL: <http://www.itesla-project.eu/>.
- Landsnet (2015). *Landsnet Performance Report 2015*. URL: http://2015.landsnet.is/wp-content/uploads/2016/05/Landsnet_USE_performance_report_2015_english-FINAL_26-5-2016.pdf (visited on 08/31/2018).
- Milano, Federico (Aug. 31, 2018). *PSAT*. URL: <http://faraday1.ucd.ie/psat.html>.
- Modelica Association (Apr. 2017). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.4*. Tech. rep. Linköping: Modelica Association. URL: <https://www.modelica.org/documents/ModelicaSpec34.pdf>.
- OSMC (Aug. 31, 2018). *OpenModelica – open-source Modelica-based modeling and simulation environment*. Ed. by Open Source Modelica Consortium. URL: <https://openmodelica.org/>.
- Siemens (Aug. 31, 2018). *PSS/E*. URL: <http://siemens.com/power-technologies/software>.
- Vanfretti, L. et al. (2016). "iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations". In: *SoftwareX* 5, pp. 84–88. DOI: 10.1016/j.softx.2016.05.001.
- Westfjord Power Company (Aug. 31, 2018). *Mjolka (Milk) River Power Station*. URL: <https://ov.is/en>.

A Appendix

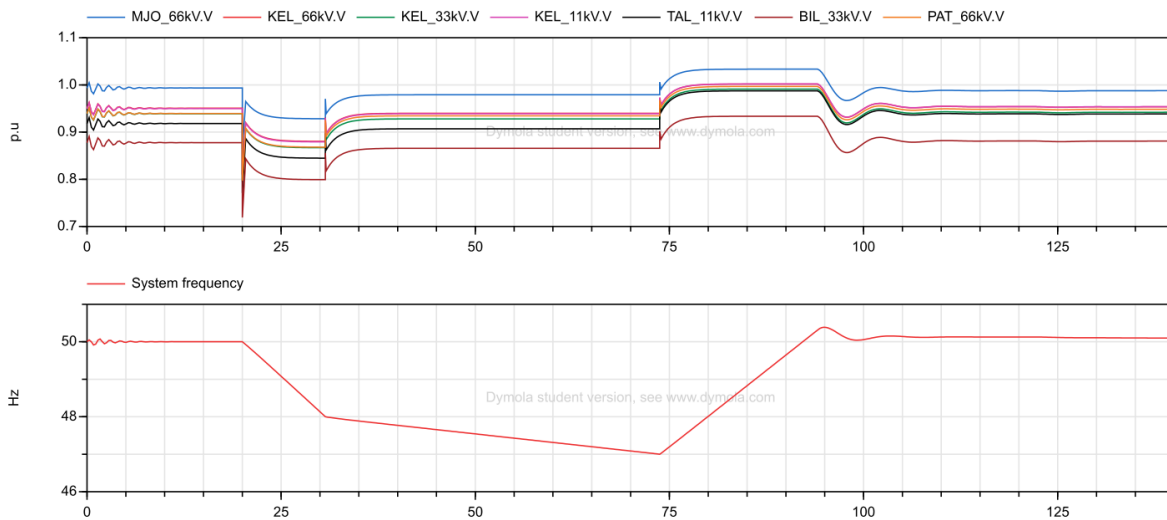


Figure 6. Voltage and frequency plots for Scenario 1.1 for the three-generator model

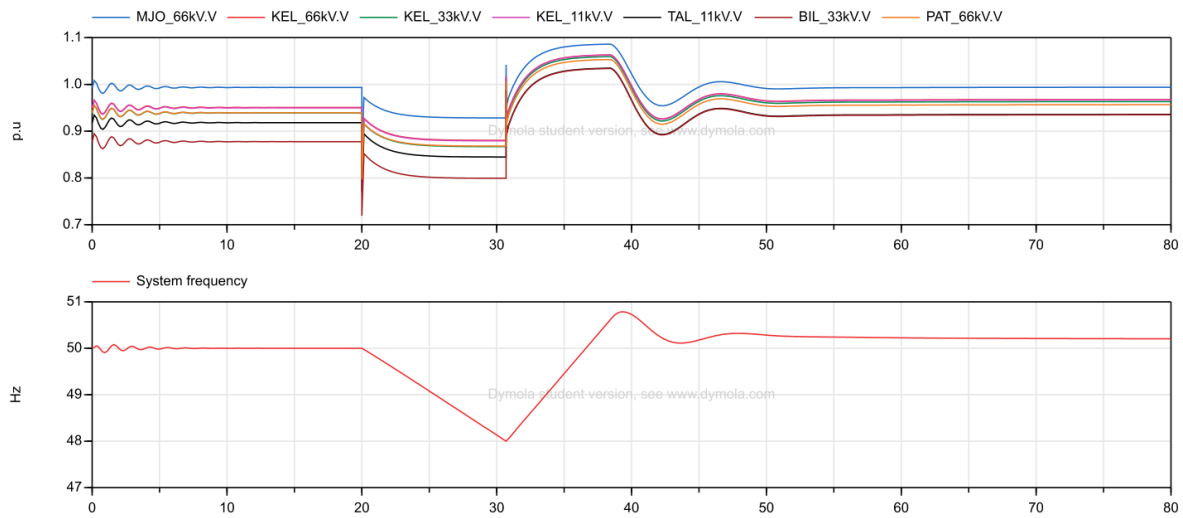


Figure 7. Voltage and frequency plots for Scenario 1.2 for the three-generator model

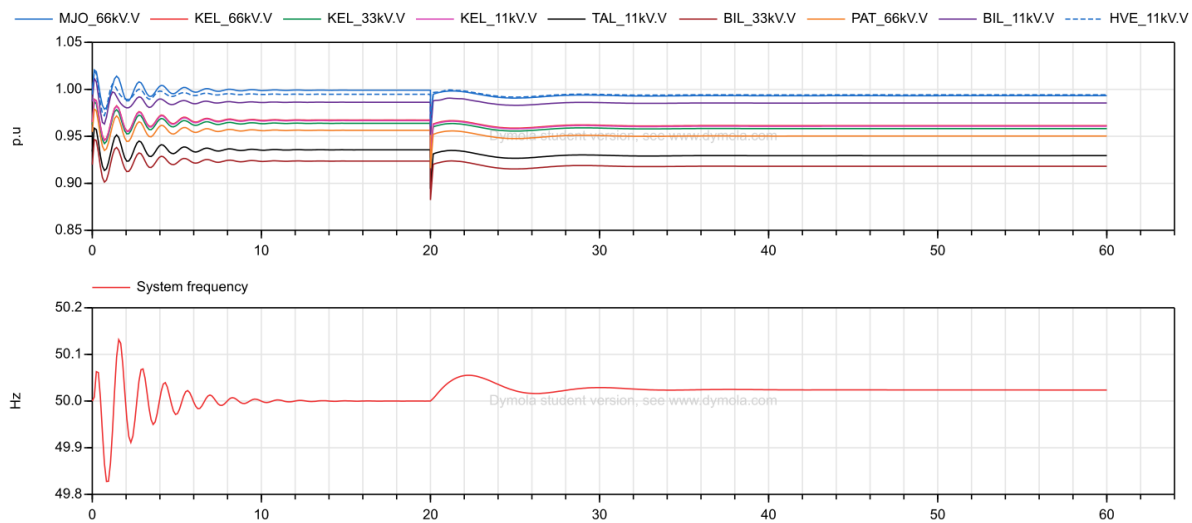


Figure 8. Voltage and frequency plots for Scenario 1 for the four-generator model

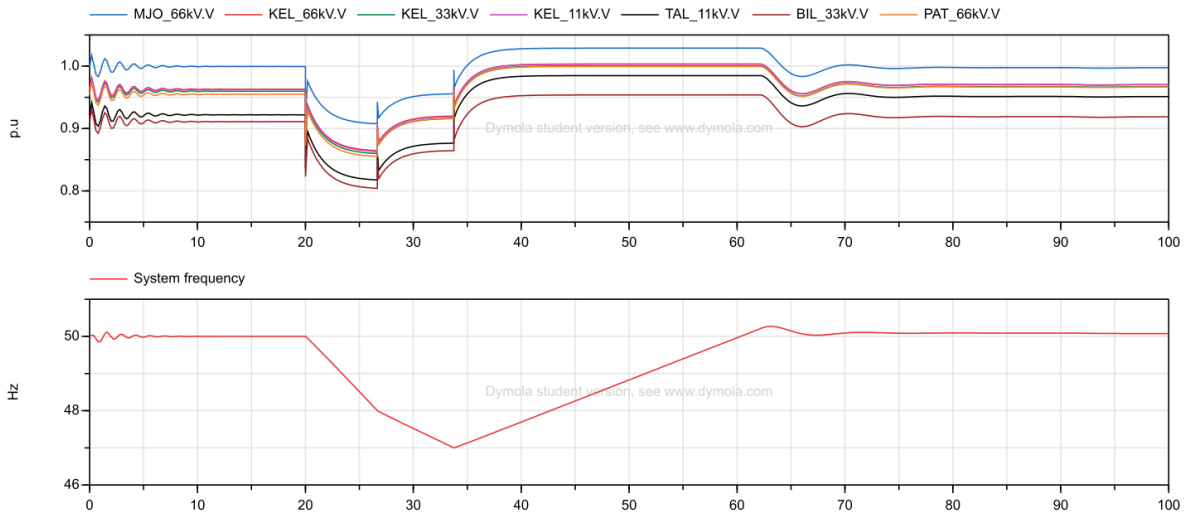


Figure 9. Voltage and frequency plots for Scenario 2.1 for the three-generator model

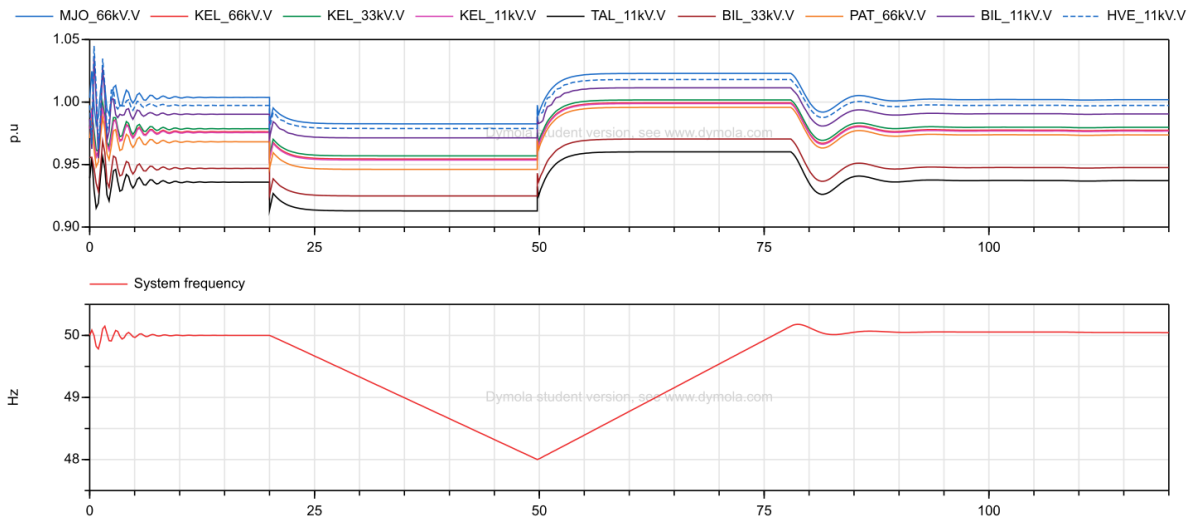


Figure 10. Voltage and frequency plots for Scenario 2.1 for the four-generator model

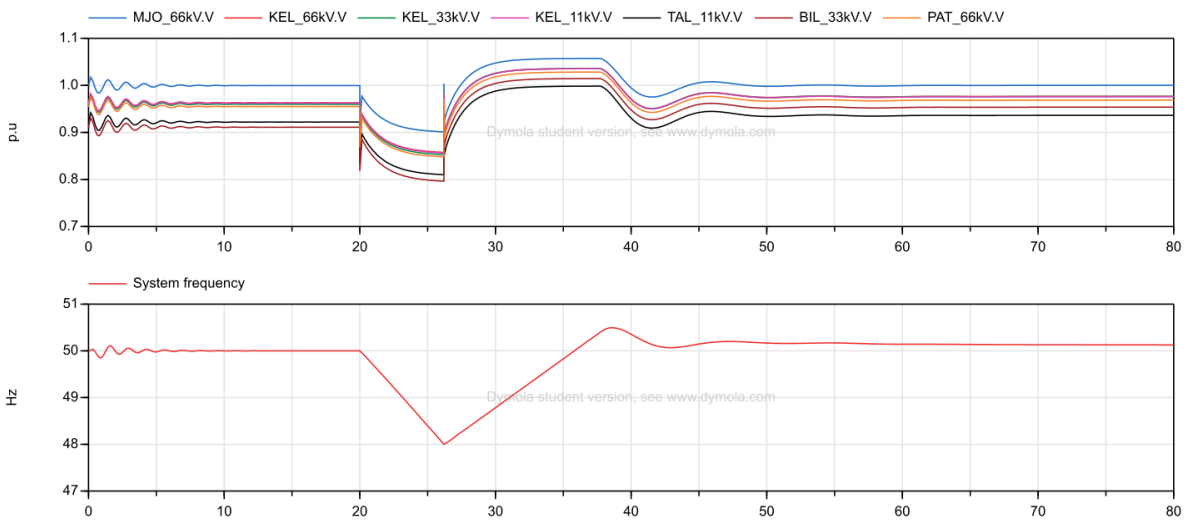


Figure 11. Voltage and frequency plots for Scenario 2.2 for the three-generator model

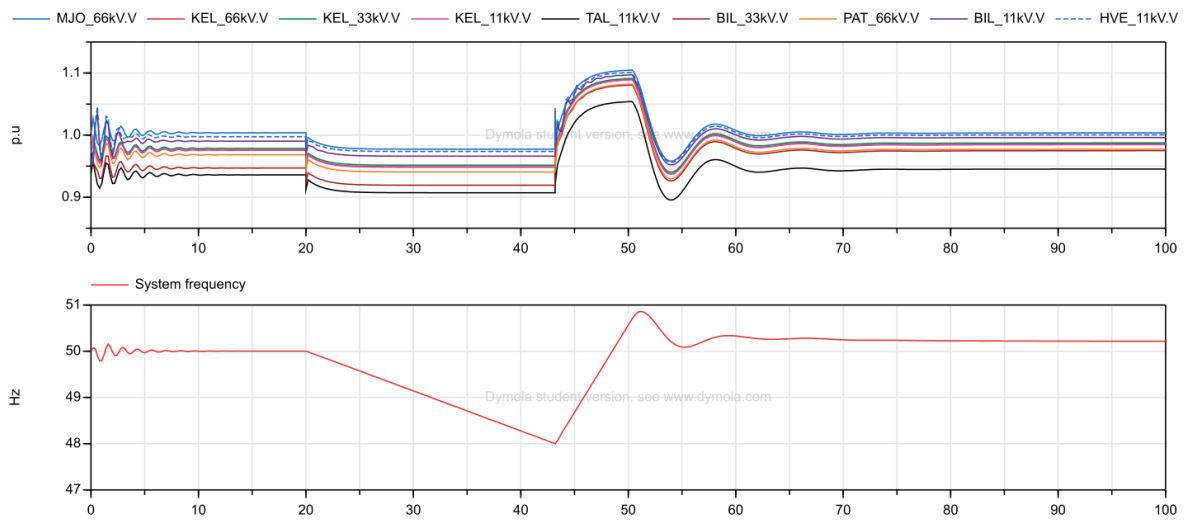


Figure 12. Voltage and frequency plots for Scenario 2.2 for the four-generator model

Modeling of PMU-Based Islanded Operation Controls for Power Distribution Networks using Modelica and OpenIPSL

Biswarup Mukherjee¹ Luigi Vanfretti²

¹Indian Institute of Technology Bombay, India, bismuk.ece@gmail.com

²Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute (RPI), USA, vanfrel@rpi.edu

Abstract

This paper describes the modeling of a frequency controller that can be applied when islanding occurs at a power distribution network with a single distributed generator. The controller function requires bus frequency measurements which, for design purposes, need to be derived (computed) during dynamic simulations. Therefore, this paper also proposes a simple new frequency computation technique that can be used during dynamic simulations. The paper also addresses a technique for stochastic modeling of load uncertainties in the time-domain using the Modelica Noise library's features. The performance of the islanded controller is evaluated under load uncertainties, different PMU (phasor measurement unit) reporting rates and communication latencies.

Keywords: frequency computation, islanded controller, random load variation, PMU, distribution network, synchrophasors, Modelica, OpenIPSL

1 Introduction

1.1 Motivations

Islanded operation in power systems is required when a part of the network consisting of both loads and generation is isolated from the rest of the power grid, and generators continue to energize that isolated network (Almas & Vanfretti, 2016). Controlling the frequency in an islanded power system is a very challenging task after an islanding occurs because it requires at least one generator in the island to restore the power/frequency balance in the island while at the same time restoring its mechanical speed before being re-synchronized to the main grid (Taranto & Assis, 2012). Alternatively, if there are enough available generators in the islanded network, the generators could be used to operate the islanded portion autonomously, which is commonly referred to as a "microgrid" (Lasseter, 2002).

Other than having enough generation capacity to match the load in the island, at least one of the generators would need to be equipped with an

isochronous controller to restore the frequency of the island to normal operating frequencies. However, this would require prior knowledge on how the microgrid will be formed and to equip all potential generators both with the traditional droop function and the isochronous function; and to know when to de-activate it. This paper explores an alternative supplementary controller that could provide the same functionalities and demonstrates this concept in the simplest case, when there is only one generator present in the island.

1.2 Literature Review

It is reported in (Franc, Taranto, & Giusto, 2013) that synchrophasor-based islanding detection schemes may be able to provide fast and reliable islanding detection. To measure the frequency PMUs are proposed in (Kirkham *et al*, 2014), and controls for re-synchronization using the PMU/phasor data have been studied in (Taranto & Assis, 2012). For islanded operation, alternatively it would be equally attractive to propose a controller capable of using the frequency estimated by PMUs. This paper proposes to use PMU measurements in both transmission and distribution networks to achieve similar goals.

For simulation, authors in (Milano, 2017) have proposed that the system frequency can be estimated from the center of inertia (COI) concept and a washout filter (WF) on the phase angle of bus voltage. The COI is an artificial modeling construct and in practice it cannot be used (Diez-Maroto *et al*, 2001). In this paper, an alternative new frequency computation technique is proposed.

A synchronous islanding control scheme is proposed in (Jacobsen *et al*, 2016). It uses a load sharing concept, frequency deviation and phase angle deviation from the islanded network. The measurements from a PMU are used to calculate the active power imbalance with respect to the main grid's frequency and the phase angle. However, correction in the frequency's DC bias in the islanded network was not addressed.

For islanded operation, different governor configurations for frequency control have been proposed (Mahatet *et al*, 2009). Such an approach requires an additional isochronous controller to bring

the frequency back to its nominal value when the system is islanded. Instead, this paper proposes an alternative supplementary controller that is cascaded to the speed control loop.

1.3 Contributions

The main contributions of this paper are as follows:

- Proposing a simple frequency computation technique that uses bus voltage angles within the simulated model, which is attractive for controller simulations when using the positive-sequence power systems dynamic modeling framework.
- A new supplementary islanded operation controller is proposed. The controller uses a PI function and it is modeled using a centralized control architecture that receives data from PMUs, thereby complementing existing generator control systems instead of replacing the existing ones. When activated it will retain a frequency deviation of zero when the distribution side is islanded from the main transmission grid.
- A technique to simulate random load variations using Modelica Noise library¹ (which is also integrated in Modelica Standard Library (MSL) 3.2.2²) features is proposed. The performance of the islanded controller is evaluated under time-domain load uncertainties in the distribution side of the test network.
- The performance of the proposed controller is studied considering different PMU reporting rates and data transmission delays.

The remainder of this paper is organized as follows. In Section 2 the proposed frequency computation technique is presented. Sections 3, 4 and 5 explain the modeling of the islanded operation controller, stochastic load model, and a model to implement the PMU reporting rate with data transmission delay, respectively. Section 6 describes the power system and simulation execution models. Finally, case studies are analyzed in Section 7 and conclusions are drawn in Section 8.

2 Frequency Computation

2.1 Theory

When the distribution grid is disconnected from the bulk transmission system, the bus voltage angles measured by PMUs in the distribution grid will deviate from those of the transmission grid. Phasor angle measurements are bounded to +/- 180 degrees. Therefore, if the bus frequency is calculated from the bus angle directly, angular measurement unwrapping will create spikes that corrupt the actual frequency

deviation. To overcome this issue and to provide useful frequency signals for control a simple method is proposed. Let V_i and V_r represent the imaginary and real parts of complex bus voltage, then the bus angle (θ) can be calculated from these two values as

$$\theta = \tan^{-1} \frac{V_i}{V_r} \quad (1)$$

Let ω be frequency of the bus voltage, then the first order derivative of the bus angle represents the frequency deviation at the bus. Therefore, the bus frequency can be represented as,

$$\omega = \dot{\theta} = \frac{V_r \dot{V}_i + V_i \dot{V}_r}{V_r^2 + V_i^2} \quad (2)$$

Equation (2) is used for implementation in Modelica.

2.2 Implementation

Figure 1 shows the block diagram used for implementation of the Modelica model of the proposed frequency computation technique.

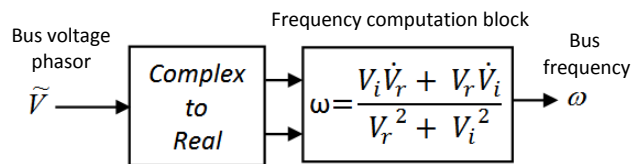


Figure 1. Block diagram for implementation of the frequency computation technique.

To compute the bus frequency the following Modelica code has been used in the “Frequency computation block” shown in Figure 1.

```
model frequencyCalculationBlockCode
  Modelica.Blocks.Interfaces.RealInput u;
  Modelica.Blocks.Interfaces.RealInput u1;
  Modelica.Blocks.Interfaces.RealOutput y;
  Modelica.Blocks.Continuous.Derivative
  derivative;
  Modelica.Blocks.Continuous.Derivative
  derivativel;
equation
  y = (u*(derivativel.y) + u1*(derivative.y))/
  ((u^2) + (u1^2));
  connect(u1, derivativel.u);
  connect(u, derivative.u);
end frequencyCalculationBlockCode;
```

The real inputs v_r and v_i represent the real and imaginary parts of the complex bus voltage and the real output y represents the calculated bus frequency.

The Modelica code uses a derivative block from Modelica Standard Library (MSL), which is shown below. This block defines a transfer function between the input u and output y . In the frequency computation

¹Online at: <https://github.com/DLR-SR/Noise>

²Online at: <https://github.com/modelica/ModelicaStandardLibrary>

the derivative block uses a gain value of $k=1$ and a time constant of $T=0.01$ sec.

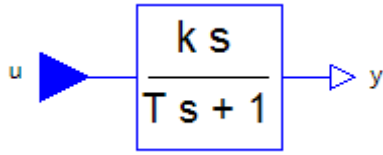


Figure 2. Model of the derivative block from the Modelica Standard Library (MSL).

2.3 Numerical results and comparison

Conventional power system tools, e.g. PSS/E (Siemens AG, 2018), compute bus frequencies using the approach shown in Figure 3. It passes the bus voltage angle through a derivative computation and a first order filter. It has been shown in (Milano, 2017) that the approach is prone to numerical problems, although it is the standard de-facto approach. An alternative method in (Milano, 2017), is only suitable for domain specific power system tools and not general-purpose ones, e.g. Dymola or OpenModelica.

To illustrate the need for the implementation proposed in this paper, the standard approach, i.e. wash-out (WF) filter, is compared with the one proposed in this paper. The Modelica implementation of the WF filter is shown in Figure 3, while simulation results are plotted in Figure 4. The parameter used for the WF filter are $k=1$, $T_f=1$ sec and $T_w=2$ sec, where T_f and T_w are the time constants of the derivative and first order filter blocks, respectively. The gain k is the same for both derivative block and first order filter. In the simulation the input angle is switched from $-\pi$ to $+\pi$ with a period of 1 sec. to mimic angle wrapping.

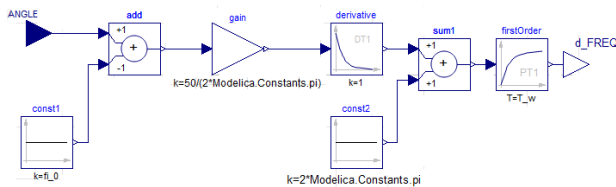
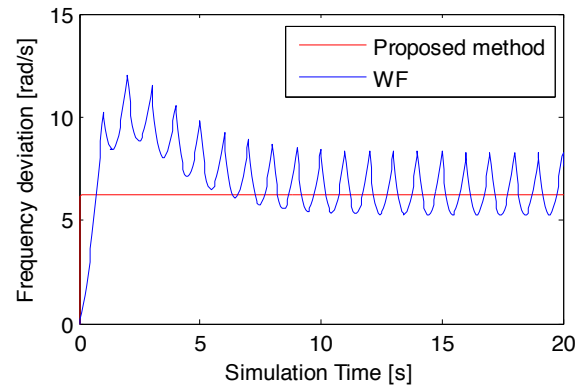
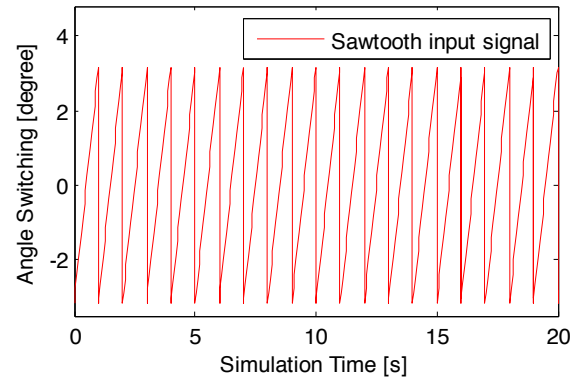


Figure 3. Washout filter (WF) implementation in Modelica.

The Figure 4a shows the traces of the proposed implementation showing $\Delta f = 0$ for angle switching in Figure 4b through $\pm\pi$ switching, while the blue trace shows the output of the WF filter. The expected output is a frequency value of 1Hz, consequently, two major issues with the WF filter can be observed: (1) the filter's response due to initialization and (2) numerical switching due to the sawtooth's input. Hence instead of using the traditional frequency computation approach, the islanded controller in the next section will use the proposed method for frequency computation.



(a) Proposed method vs WF model



(b) Input signal

Figure 4. Comparison between the proposed frequency computation technique and the WF model.

3 The Islanded Operation Controller

3.1 Islanded operation control function

A schematic of the proposed islanded controller model is shown in Figure 5.

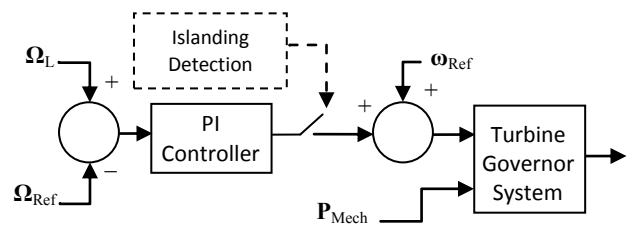


Figure 5. Schematic of the proposed controller.

The controller is activated when it detects that the distribution network is islanded from the transmission grid. In the schematic the error signal is obtained from the load bus frequency (Ω_L) and the reference frequency (Ω_{Ref}). The output of this controller, along with the reference speed (ω_{Ref}), provide the new input error signal of the speed control loop in the governor system. P_{Mech} represents the mechanical power input to the turbine corresponding to a prescribed power dispatch.

3.2 Modelica Implementation

The Modelica implementation of the proposed controller is shown in Figure 6. The proposed controller is highlighted with a dotted line surrounding it, while the GENSAL block corresponds to the synchronous generator; IEEEESGO corresponds to the gas and turbine model, and SEXS to the excitation control system of the generator. The overall system frequency is varied by introducing a speed change in the governor system of the transmission side generator model G1 in Figure 9 and 10.

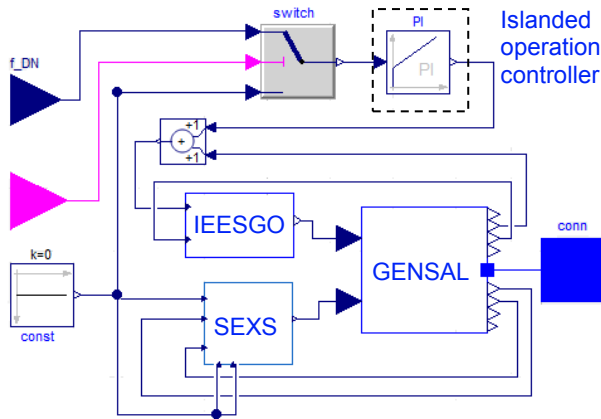


Figure 6. Modelica implementation for the distribution side generator model (G22) with the islanded operation controller, see Figure 9 for the network model.

The IEEE Standard for Interconnecting Distributed Resources with Electric Power Systems (IEEE Standard, 1547.2-2008) states that the DGs (Distributed Generations) must be disconnected from the isolated grid within 2 s after an unintentional islanding event. However, the goal of the proposed controller is to avoid DG disconnection and operate the grid autonomously. Hence, instead of tripping the generator, when the distribution side frequency reaches tripping thresholds, the trip signal goes to the breaker to island the distribution side, and the same time an activation signal ‘start_islanding’ activates the islanded operation controller. The PI controller’s output can be expressed as in the Table 1, where K_p and K_i represent the proportional and integral gain of the islanded operation controller respectively.

Table 1. Output truth table of voltage controller

| Boolean signal (start_islanding) | Output of islanded controller (y) |
|----------------------------------|---|
| True | $y = \Omega_L * \left(K_p + \frac{K_i}{S} \right)$ |
| False | 0 |

4 Stochastic Load Modeling

The Modelica Noise Library allows users to model stochastic behavior, and it can be used along with the load model with external input of OpenIPSL (Baudette *et al*, 2018) under, `OpenIPSL.Electrical.Loads.PSSE.Load_ExtInput`, to model the load uncertainties in any power network. Here, a white noise has been injected to the load model. Note that white noise generates a signal having normal distribution characterized by a mean and variance.

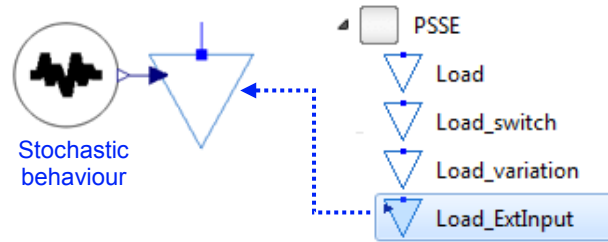


Figure 7. Stochastic model to simulate load variation in a power network.

5 Modeling of PMU Reporting Rates and End-to-End Delay

To Model the reporting rate of a PMU device, a Zero Order Hold (ZOH) block from Modelica Standard Library can be used to simulate different data “resolutions”, i.e. different reporting rates, streamed by a PMU device. The time delay due to data transmission from a PMU to Phasor Data Concentrator (PDC) has been modeled using the ‘fixedDelay’ block from the MSL that uses the following Modelica code shown below. Note that it uses the Modelica ‘delay’ operator, which is a unique Modelica language feature. This operator introduces a fixed time delay between a real input and a real output signal.

```
block FixedDelay "Delay block with fixed DelayTime"
  extends
  Modelica.Blocks.Interfaces.SISO;
  parameter SI.Time delayTime(start=1)
  "Delay time of output with respect to input signal";
  equation
    y = delay(u, delayTime);
end FixedDelay;
```

The implementation of the reporting rate and delay block is shown below, and the PMU reporting rate is shown in Figure 8. In this paper the performance of the controller has been studied for both the PMU reporting rate and delay due to data transmission by varying the sampling period of the ZOH block and the delay time of the ‘fixedDelay’ block.

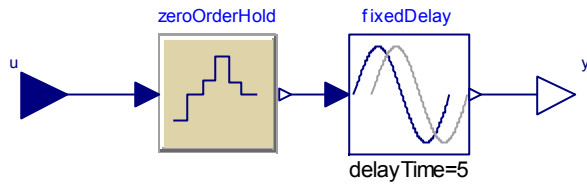


Figure 8. Modeling of PMU reporting rate and data transmission delay.

6 Power System and Simulation Execution Models

6.1 Power System Model

Figure 9 shows the power system model used for analysis. It is comprised by a transmission network and a distribution network. The circuit breakers CB1 and CB2 are controlled using logic equations implemented in a simulation set-up block, which is discussed next.

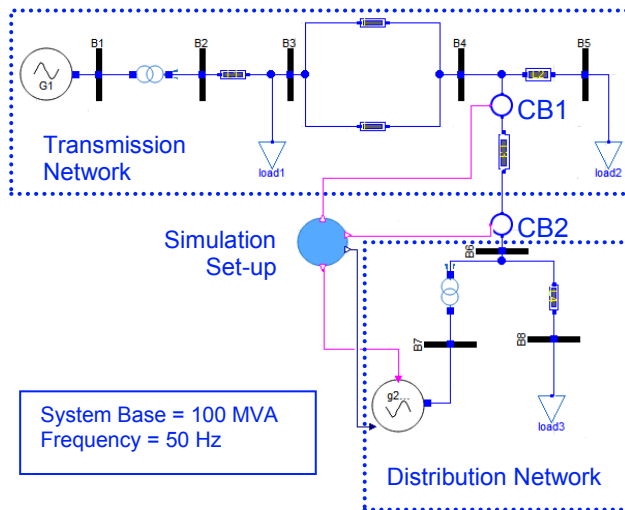


Figure 9. Modelica model of the test power system.

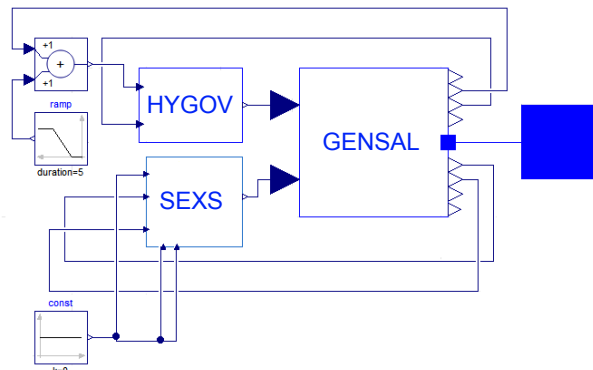


Figure 10. Generator model G1 in the transmission portion of the network.

6.2 Simulation Set-up Implementation

The implementation of the simulation set-up block is shown in Figure 11. It is used to create the islanding event and to activate the islanded operation controller. A ramp signal is activated in Glat $t=6$ seconds when the simulation starts and lasts for 5 seconds to vary the frequency. The 'Frequency computation block' calculates the frequency for the distribution side network from B6's bus voltage.

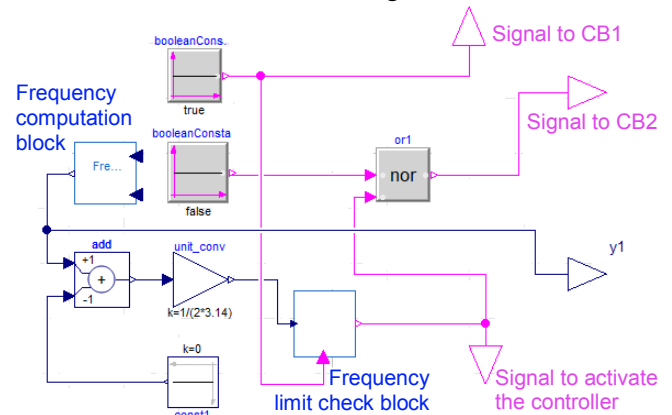


Figure 11. Modelica model of the simulation set-up block.

A true Boolean signal is sent to the circuit breaker CB2 when the frequency limit is reached. This condition checks the frequency deviation to the set-point limit set in the 'Frequency limit check' block. The output of this block is also used to activate the proposed islanded controller, while a Boolean true signal keeps the circuit breaker CB1 closed in the transmission side network while maintaining the transmission line energized. Regardless, CB1 is modeled in such way that it can be closed and open as CB2, which will be used in further studies. If the frequency limits are provided in nominal frequency values instead of a frequency difference, i.e. 49.95 and 50.05 Hz instead of ± 0.05 Hz, the block 'const1' can be used to introduce a 50 Hz offset.

7 Case Studies

In the following case studies, except in Case 1, a steam turbine and governor system are used in the model of the distribution side generator G22 to analyze the performance of the proposed islanded operation controller. For all case studies, the same basic simulation set-up described in the previous section is used; hence, the disturbance applied corresponds to the ramp input into the governor reference as shown in Figure 10.

7.1 Case 1:

The feasibility of using proposed controller in either hydro or gas turbines is studied in this case. This is

necessary as DER's include small hydro units, thermal and gas-power sources. In power systems, HYG0V is used to model hydraulic turbine and governor systems, while IEEESGO can be used to model steam turbine and governor systems.

The test system responses due to the controller's action were analyzed by plotting the frequency deviation in the distribution side network using both the HYG0V and the IEEESGO turbine-governor systems. Figure 12 shows that the distribution side frequency deviation is zero when the proposed PI controller is activated, whereas a steady state error is present when there is no such control action regardless to the turbine-governor type.

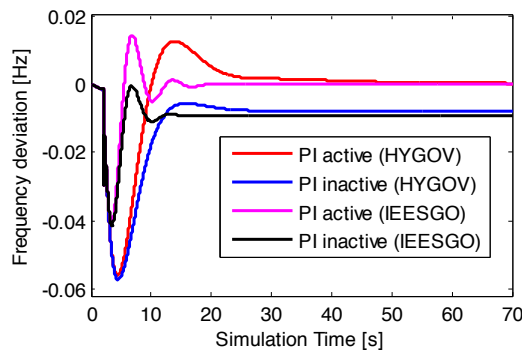


Figure 12. Case 1 - Frequency deviation for different turbine-governor systems.

In the case of the IEEESGO turbine-governor system the maximum instantaneous values of frequency deviations are 0.0414 Hz and 0.0405 Hz respectively when the controller remains inactive and active. However In case of HYG0V turbine-governor systems the maximum instantaneous values are 0.057 Hz (when control action remains inactive) and 0.055 Hz (when control action remains active).

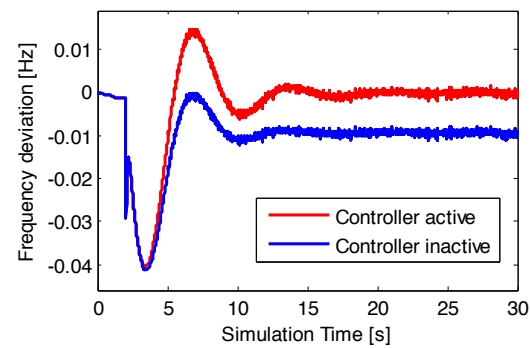
Observe that the responses of HYG0V results from a larger frequency transient which is due to the non-minimum phase characteristic of the hydro turbine. To minimize this transient or reduce it to allowable operational limits it is necessary to redesign the governor control system, which will be discussed in a future paper.

7.2 Case 2:

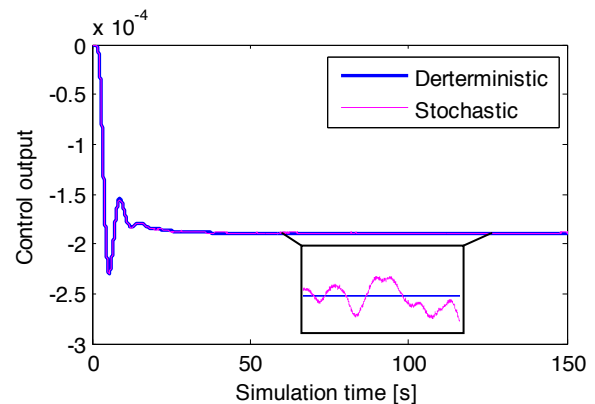
Here the controller's response has been analyzed by plotting the frequency deviation of the distribution network when the load in B6 has a noise level with standard deviation (s.d.) of 0.0001. The results are plotted in Figure 13a. As it can be observed, the modeling of load uncertainties allows to evaluate the controller's effort during the islanding and also when normal operating conditions have been reached.

Figure 13b helps to show the impact of stochastic loads. It shows the islanded operation control output

error for both deterministic and stochastic responses. Note from Figure 13b that deterministic load models do not allow to accurately capture the controller's response due to time varying load changes. The ability to capture this behavior can allow to create controls that minimize the impact of stochastic variations on the turbine, which will be subjected to further work.



(a) Frequency deviation



(b) Controller output

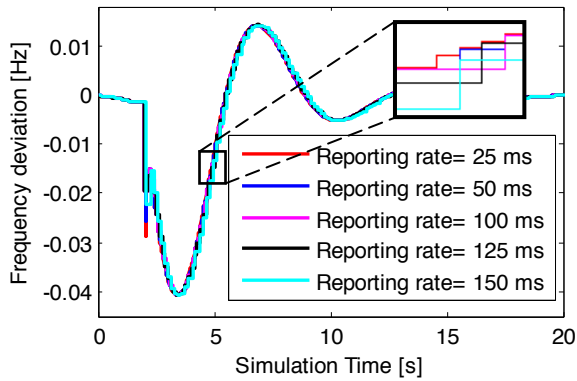
Figure 13. Case 2 – Stochastic and deterministic model responses.

7.3 Case3:

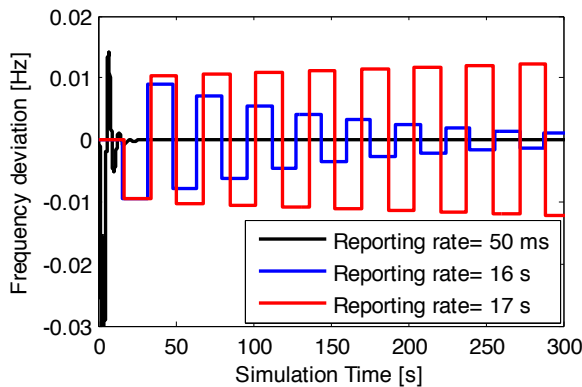
This case study has been carried out to analyze the performance of the islanded operation controller for the test network in Figure 9 considering the impact of the PMUs reporting rate. The frequency deviation has been plotted for different sampling periods (ZOH) when the islanded controller is active. The plots for this case study are shown in the Figure 14a.

From Figure 14a observe that delays from 25 to 150 ms have no major impact on the controller's performance, this is because the frequency dynamics being controlled are much larger than typical PMU reporting rates. However as shown in Figure 14b, when the reporting rate is set to tens of seconds, the control loop becomes unstable i.e. for the reporting rate > 16 s. This is a positive result, as typical PMU reporting rates are ≤ 16 s, i.e. 10, 30, 50, 60 samples per seconds.

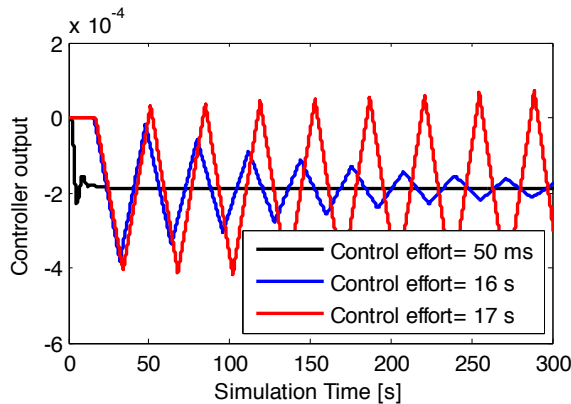
Although the lowest PMU reporting rates is of 10 samples per second, simulations have been carried out with much slower reporting rates to determine the stability margin of the controlled system.



(a) Frequency deviation



(b) Frequency deviation



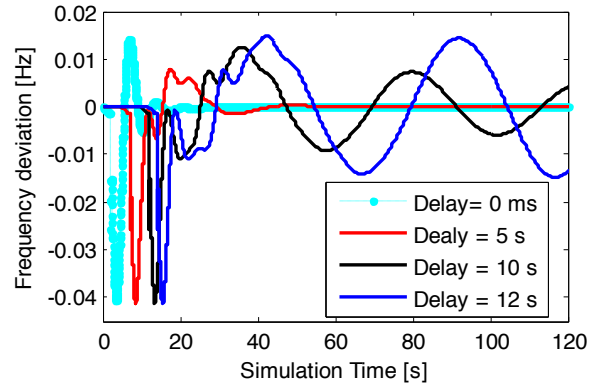
(c) Controller output

Figure 14. Case 3 – Analysis of different PMU reporting rates.

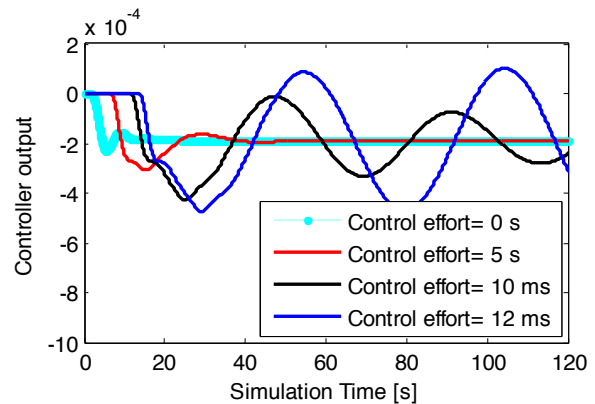
7.4 Case 4:

This case study analyses the impact of data transmission delay. A ‘fixedDelay’ block is used to mimic the aggregate time-delay from a PMU device to the controller. The results are shown in Figure 15a and 15b.

As it can be observed in Figure 15a that the maximum delay bound is time delay (t_d) \approx 12s. These results are encouraging, as typical synchrophasor systems only incur in delays in the order of a 100s of milliseconds, up to a few seconds, and thus, delay compensation will not be critical as in other PMU based controls (Almas & Vanfretti, 2016).



(a) Frequency deviation

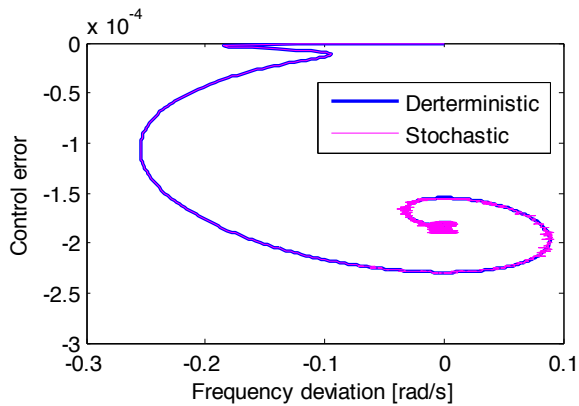


(b) Controller output

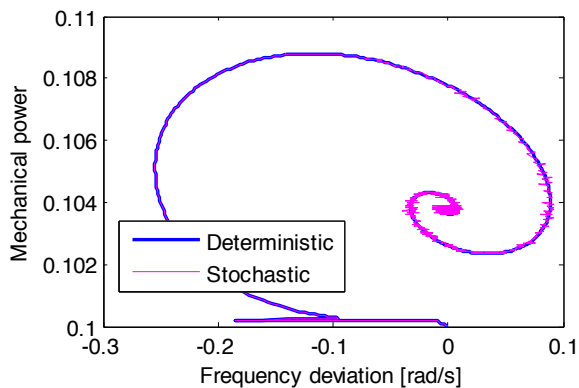
Figure 15. Case 4 - Controller output for different delays.

7.5 Case 5:

This case study analyses the impact of frequency deviation over both control error and mechanical power output of the turbine governor system. From Figure 16a it can be observed that for both deterministic load and stochastic load model the control error decreases up-to 0.023 %. This shows the impact of modeling the stochastic behavior of the load for control design. Meanwhile, Figure 16b shows the plot of frequency deviation for mechanical power output of the turbine governor system for both stochastic and deterministic load models. Observe that with the increase of the frequency deviation, the mechanical power increases up-to 10.88 %. This shows that stochastic load modeling is necessary when analyzing turbine-governor control systems.



(a) Frequency deviation vs control error plot(see Figure 13b for the time domain plot of the control error)



(b) Frequency deviation Vs mechanical power

Figure 16. Case 5 – Impact of stochastic load modeling in turbine-governor system control performance.

8 Conclusions

The following conclusions can be drawn from the work presented in this paper:

- The proposed frequency computation provides better results than the traditional WF filter in case of angle wrapping from +/- 180 degrees.
- To simulate random load variations in any power network the Modelica Noise Library features is combined with OpenIPSL is a feasible solution that helps to capture the impact of time domain load/generation uncertainties in the controller performance.
- The proposed supplementary islanded frequency controller can be activated to retain a frequency deviation of zero when islanding occurs. It also performs satisfactorily to correct the frequency deviations when subjected to load uncertainties.
- The current PMU report rates and typical delays that synchrophasor systems experience will not have a major impact on the controller performance.

Further work will be to investigate the performance of this controller under multiple realizations of load uncertainties (noise level) in order to analyze the impact of uncertainties on the system.

Reproducibility of Research

The models used to obtain the results in this paper are available online on the following Github repository: https://github.com/ALSETLab/2018_AmericanModelicaConf_PMUBasedIslanding

Acknowledgements

The authors would like to acknowledge former members of the now extinct SmarTS Lab research group for their valuable feedback and comments towards building the models for this work, especially Tin Rabuzin.

Author B. Mukherjee was supported by Explora'Sup grant (Auvergne Rhône Alpes Scholarship) from Grenoble Institute of Technology (Grenoble-INP), for the year of 2016/2017.

The work of L. Vanfretti was also supported in part by the Engineering Research Center Program of the National Science Foundation and the Department of Energy under Award EEC-1041877 and in part by the CURENT Industry Partnership Program.

References

- Almas, M.S. & Vanfretti, L., "RT-HIL Implementation of the Hybrid Synchrophasor and GOOSE-Based Passive Islanding Schemes," *IEEE Transactions on Power Delivery*, vol. 31, pp. 1299-1309, June 2016.
- Taranto, G.N., Assis, Tatiana M.L. "Automatic Reconnection From Intentional Islanding Based on Remote Sensing of Voltage and Frequency Signals," *IEEE Transactions on Smart Grid*, December 2012, Vol. 3, pp. 1877-1884
- Lasseter, R.H., "MicroGrids," *Power Engineering Society Winter Meeting, 2002. IEEE*, vol.1, no., pp.305,308 vol.1, 2002
- Franc, R., Sena, C., Taranto, G.N., & Giusto, A., "Using synchrophasors for controlled islanding-A prospective application for the Uruguayan power system," *IEEE Trans. Power Syst.*, vol. 28, no. 2, pp. 2016-2024, May 2013
- Kirkham, H., Dagle, J., Sun, Y., *PMU Measurement Technology*. s.l.: <https://certs.lbl.gov/sites/all/files/kirkham-pmu-measurement-technology.pdf>, 2014. p. 10
- Milano, F., and Ortega, A., "Frequency divider," *Power & Energy Society General Meeting, 2017 IEEE*
- Diez-Maroto. L., Vanfretti. L., Almas, M.S., Jonsdottir. G.M., Rouco.L, "A WACS exploiting generator Excitation Boosters for power system transient stability enhancement," *Elsevier-Electric Power Systems Research*. <https://doi.org/10.1016/j.epsr.2017.03.019>
- Jacobsen, M.R., Laverty, D., Best, R. J., "A laboratory experiment of single machine synchronous islanding using PMUs and Raspberry Pi – A platform for multi-machine islanding," *Power & Energy Society General Meeting, 2016*

PukarMahat, Zhe Chen, and BirgitteBak-Jensen, “Gas turbine control for islanding operation of distribution systems,” *Power & Energy Society General Meeting, 2009 IEEE*

Modelica Noise. [Online] <https://github.com/DLR-SR/Noise>

Modelica Standard Library, [Online] <https://github.com/modelica/ModelicaStandardLibrary>

Siemens AG. (2018). PSS®E – high-performance transmission planning and analysis software. Retrieved from:

<https://www.siemens.com/global/en/home/products/energy/services/transmission-distribution-smart-grid/consulting-and-planning/pss-software/pss-e.html>

IEEE Application Guide for IEEE Standard for Interconnecting Distributed Resources With Electric Power Systems, *IEEE Std. 1547.2-2008, 2008*

Baudette, M., Castro, M., Rabuzin, T., Lavenius, J., Bogodorova, T., & Vanfretti, L. (2018). OpenIPSL: Open-Instance Power System Library – Update 1.5 to “iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations”, *SoftwareX*. <https://doi.org/10.1016/j.softx.2018.01.002>

OpenIPSL GitHub, [online] <https://github.com/OpenIPSL/OpenIPSL> .

Modelica Documentation, [Online] [https://build.openmodelica.org/Documentation/ModelicaReferenceOperators.%27delay\(\)%27.html](https://build.openmodelica.org/Documentation/ModelicaReferenceOperators.%27delay()%27.html)

Almas, M.S. & Vanfretti, L., “Impact of time-synchronization signal loss on PMU-based WAMPAC applications,” *IEEE Power and Energy Society General Meeting (PESGM)*, pp.1 – 5, 2016

ModestPy: An Open-Source Python Tool for Parameter Estimation in Functional Mock-up Units

Krzysztof Arendt¹ Muhyiddine Jradi¹ Michael Wetter² Christian T. Veje¹

¹Center for Energy Informatics, University of Southern Denmark, Denmark, {krza,mjr,veje}@mmmi.sdu.dk

²Lawrence Berkeley National Laboratory, USA, mwetter@lbl.gov

Abstract

The paper presents an open-source Python tool for parameter estimation in FMI-compliant models, called *ModestPy*. The tool enables estimation of model parameters using user-defined sequences of methods, which are particularly helpful in non-convex problems. A user can start estimation with a chosen global search method and subsequently refine the estimates with a local search method. Several methods are available already and the tool's architecture allows for easily adding new ones. The advantages of having a single interface to multiple methods and using them in sequences are highlighted on a case study in which the parameters of a Modelica-based gray-box model of a building zone (nonlinear, multi-output) are estimated using 9 different combinations of methods. The methods are compared in terms of accuracy and computational performance.

Keywords: FMI, parameter estimation, Python, open-source

1 Introduction

1.1 Background

The Functional Mock-up Interface (FMI) is becoming a *de facto* standard co-simulation interface, as of 2018 being supported by over 100 simulation tools (<http://fmi-standard.org/tools/>). The support of the FMI in simulation tools varies from full support, especially in Modelica tools, to a subset of FMI 1.0 / 2.0, import / export, co-simulation / model exchange. Models compliant with FMI are referred as Functional Mock-up Units (FMUs).

Using FMUs allows for a flexible co-simulation between models developed in different software environments. It is also attractive for resource-limited embedded systems with no need to install a GUI-based simulation environment, or because relying on FMI makes the system less dependent on specific tools and vendors. In addition, the common interface to models developed in different software environments makes it possible to develop generic tools for co-simulation, system identification, or optimization. System identification methods are commonly used to calibrate models with respect to the real system. In the case of FMUs, the model structure is typically already defined in the FMU, so the problem is nar-

rowed down to the estimation of model parameters and/or states. In fact, several FMI-specific tools for parameter and state estimation have been developed in recent years.

Bonilla et al. (2017) developed a GUI tool for static (batch) parameter estimation in FMUs (compliant with Model Exchange 2.0), based on a global-search Multi-Objective Genetic Algorithm (MOGA). The main motivation for the development of this tool was to facilitate the coupling of different modeling languages, tools and optimization algorithms, while being customizable. The authors stated that the implementation of additional optimization algorithms is planned.

Bonvini et al. (2014) developed a state and parameter estimation Python tool based on Unscented Kalman Filter (UKF), compliant with Model Exchange 1.0. The UKF is a recursive estimation method, meaning it is suitable for on-line applications where the states or parameters need to be continuously updated. The authors presented the capabilities of the tool on a fault detection and diagnosis (FDD) case study, in which it was used to identify a faulty valve in a simple theoretical system. In another exemplary application, the tool has been used for an on-line estimation of the number of occupants in a building based on indoor temperature and CO₂ (Sangogboye et al., 2017).

Vanfretti et al. (2016) developed the Rapid Parameter Identification toolbox (RaPIId), used for parameter estimation in FMUs. The tool is developed as a Matlab/Simulink plug-in and can be called using the Matlab command line interface, but is also equipped with a GUI allowing it to be used as a standalone application. Multiple optimization methods are available, such as `fmincon` from Matlab and heuristic algorithms, like a particle swarm optimization (PSO) implemented by the authors.

Kampfmann et al. (2017) proposed a work flow for parameter estimation in FMUs, based on open-source tools. The performance of the work flow was demonstrated on a real industrial problem of a three arm Delta Robot. The estimation problem is formulated using the maximum likelihood approach and the underlying optimization problem is solved using the Levenberg-Marquardt algorithm.

De Coninck et al. (2016) developed a more specialized Python toolbox for gray-box system identification for buildings. The tool automates both the model selection (out of a group of predefined models) and parameter estimation. The toolbox relies on the JModelica.org plat-

form which is used for compilation of models, simulation and optimization. The direct collocation method is used to solve the underlying optimization problem. Since the collocation method is a local optimization method, the initial guesses of parameters are either inherited from the previous estimation runs or a set of initial guesses is constructed using the Latin hypercube sampling method. The toolbox is currently in use in MPC systems in existing buildings (De Coninck and Helsen, 2016).

Andersson et al. (2012) implemented in *JModelica.org* three derivative-free optimization algorithms suitable for parameter estimation in FMUs. The algorithms include the Nelder-Mead simplex method, a differential evolution algorithm, and a genetic algorithm. The algorithms can be accessed through a Python interface.

Despite the diverse landscape of the tools and work flows presented, there are still some niche applications not specifically addressed. For example, there are no lightweight, generic, open-source tools that would enable automated testing of multiple algorithms, including algorithm sequences (for global and local search), and be easily deployable on the target machines and integrable into other codes. The available tools are either tied to specific optimization algorithms, specific proprietary platforms, or large software environments. This paper presents a new tool that potentially fills this niche.

1.2 Paper Objective

The objective of this paper is to present a new open-source Python tool for parameter estimation in FMUs, called *ModestPy* (Arendt, 2017). Through the PyFMI (Andersson et al., 2016) the tool is compatible with FMI 1.0/2.0 and Co-Simulation/Model Exchange. The main novelty of the tool compared to the ones presented is that it includes several optimization methods and enables easily running the methods in user-defined sequences.

The initial motivation of this work was to facilitate parameter estimation in models of buildings and HVAC systems for the purpose of MPC. One of the often mentioned requirements for MPC to become economically viable in buildings is an automated creation and updating (e.g. through parameter estimation) of predictive models (Rockett and Hathway, 2017). Building and HVAC models are often non-linear and in general can be non-differentiable. In this context, *ModestPy* is a generic tool with no assumptions on the model structure, offering a high flexibility in terms of combining different algorithms. Since the performance of specific methods is model-dependent, the aim of the tool is to support multiple algorithms. Currently, the tool supports two algorithms implemented by the authors (genetic algorithm and pattern search) in addition to several algorithms from the *SciPy* eco-system (Jones et al., 2001).

2 Software Description

ModestPy is designed with the ease of use and installation in mind. It is compatible with both Python 2.7 and 3 and

was tested on Windows and Linux. The package can be installed using the command `pip install modestpy`.

The package structure is modular (Fig. 1). A user needs to interact only with one class, `Estimation` from `estimation.py`, and its two methods: `estimate()` and `validate()`.

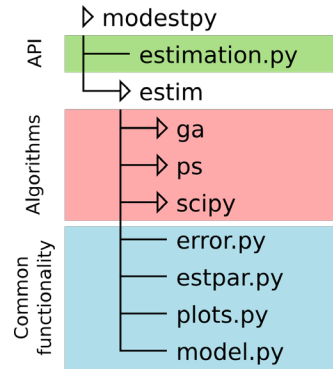


Figure 1. Package structure (testing, logging and auxiliary modules excluded for clarity).

The implemented algorithms are kept in separate modules under `modestpy.estim`. The algorithms share some common functionality covering the error calculation (`error.py`), estimation parameter interface (`estpar.py`), plotting (`plots.py`), and FMU model interface (`model.py`). The interaction with FMUs is provided by *PyFMI* (Andersson et al., 2016). Two types of error metrics are implemented, the total mean-square error (MSE_{tot}) and the total normalized mean-square error ($NMSE_{tot}$), calculated as follows:

$$MSE_{tot} = \sum_i \frac{\sum_{t=1}^N (\hat{Y}_i^t - Y_i^t)^2}{N}, \quad (1)$$

$$NMSE_{tot} = \sum_i \frac{MSE_i}{\bar{Y}_i^2}, \quad (2)$$

where \hat{Y}_i^t is the measured value of variable i at time step t , Y_i^t is the simulated value of variable i at time step t , \bar{Y}_i is the mean measured value of variable i , N is the number of time steps, and MSE_i is the mean-square error for variable i . Eq. (1) is suggested for single-output models ($i = 1$) or models with physically comparable outputs. Eq. (2) is suggested for multi-output models ($i > 1$) with physically incomparable outputs, e.g. temperature and CO_2 . It should be noted that \bar{Y}_i in Eq. (2) needs to be non-zero. Other norms can be easily implemented in `error.py` if needed.

2.1 Algorithms

A user can estimate parameters using a single algorithm or an arbitrarily designed sequence of algorithms. The sequences typically contain two methods, the first for global search and the second for local search, but if needed it is possible to queue more methods, e.g. multiple genetic algorithms with different evolution parameters (best solution from each run is saved and propagated to the next

one). It is, therefore, possible to quickly test and evaluate different methods and combination of methods on a given problem.

Currently two in-house algorithms are implemented, a genetic algorithm (GA) and a generalized pattern search (GPS), in addition to the interface to several well-known algorithms from the *SciPy* eco-system (Jones et al., 2001), specifically those compatible with the function `scipy.optimize.minimize()`. The *SciPy*'s algorithms tested by the authors with results presented in this paper include the sequential least squares programming (SLSQP) (Kraft, 1988), the limited memory Broyden-Fletcher-Goldfarb-Shanno with box constraints (L-BFGS-B) (Byrd et al., 1995), and the truncated Newton method (TNC) (Nash, 1984).

The implemented GA (Algorithm 1) is based on the standard operations of tournament-based selection and crossover. The mutation is adaptive and depends on the population diversity, defined as the ratio of shared genes among the individuals. If the population is diverse, a standard mutation is applied in which a low mutation rate (10% by default) is used, but genes can mutate within a wide range (entire parameter range by default). If the population diversity is low, the population is split into 1/3 and 2/3 parts. The larger group (2/3) undergoes the standard mutation (low mutation rate, wide range of possible values), and the smaller group undergoes mutation with an increased rate (33% by default), but the genes are allowed to change only within a small range (equivalent to a stochastic local search). By tuning $mutrate_1$, $mutrate_2$, $range_1$, and $range_2$ in Algorithm 1, a desired balance between the exploratory and local search of the evolution can be found. The algorithm continues until the error stops decreasing or the maximum number of generations is reached. The tolerance criterion checking can be delayed by a user-defined number of generations (10 by default), allowing for continuing the evolution for some time even if the subsequent generations do not improve the solution. GA is a metaheuristic global search algorithm, which is often able to deal with complex non-convex functions, but does not guarantee that the global optimum is found. In fact, the solution might be worse than in the case of local search methods. It is, therefore, often coupled with some local search methods, as GPS, least squares or Newton-based methods.

The implemented GPS (Algorithm 2) is a simple gradient-free local search method. The algorithm requires an initial guess \mathbf{x}_0 and an initial step size s . Thereafter it starts a series of orthogonal exploratory moves, by changing one parameter at a time (in both positive and negative directions) and evaluating the cost function $f(\mathbf{x})$. The parameter vector propagated to the next iteration is the one with the lowest cost function value. If the given step size does not yield a reduction in the cost function value, it is reduced by a factor of 2 and the procedure is repeated. The algorithm stops when the solution does not improve despite k_{max} reductions of the step size. GPS usually re-

Algorithm 1 Genetic algorithm implemented in *ModestPy*

```

Initialize population  $pop_1$  with  $N$  individuals
while generation  $g < g_{max}$  and error decreasing do
  for all individuals in  $pop_1$  do
    Evaluate cost function
  end for
  Initialize empty population  $pop_2$ 
  Add fittest individual from  $pop_1$  to  $pop_2$  {elitism}
  for  $i = 0$  to  $N - 1$  do
     $ind1 \leftarrow$  tournament( $pop_1$ )
     $ind2 \leftarrow$  tournament( $pop_1$ )
     $child \leftarrow$  crossover( $ind1, ind2$ )
    Add  $child$  to  $pop_2$ 
  end for
  if  $pop_2$  is diverse then
     $pop_2 \leftarrow$  mutate( $pop_2, mutrate_1, range_1$ )
  else
     $X\%$  of  $pop_2 \leftarrow$  mutate( $mutrate_1, range_1$ )
     $mutrate_2 \leftarrow mutrate_1 K \{K > 1\}$ 
     $range_2 \leftarrow range_1 L \{1 > L > 0\}$ 
     $(100 - X)\%$  of  $pop_2 \leftarrow$  mutate( $mutrate_2, range_2$ )
  end if
   $pop_1 \leftarrow pop_2$ 
   $g \leftarrow g + 1$ 
   $\mathbf{x} \leftarrow$  parameters of fittest individual
end while
return  $\mathbf{x}$ 

```

quires more iterations than gradient-based methods, but the method can deal with models that are not differentiable or continuous. A similar algorithm is implemented in the generic optimization program GenOpt (Wetter, 2001).

2.2 Usage

ModestPy does not have a GUI (although there are plans to develop one in the future) and is aimed to be used directly in Python scripts. First, the user has to instantiate the `Estimation` class. All the estimation and validation settings are specified during the instantiation. The required arguments are as follows: path to the working directory (string), path to the FMU (string), data frame with the inputs (pandas DataFrame), known inputs (dictionary), parameters to be estimated (dictionary), and measured data (pandas DataFrame). The user can control other aspects of the estimation with the optional arguments, as discussed in the documentation (Arendt, 2017). Secondly, the estimation and validation methods are called to retrieve the results.

The exemplary Python code setting up an estimation using GA+GPS and using the MSE cost function is as follows:

```

from modestpy import Estimation

session = Estimation(
    workdir, # string
    fmu_path, # string

```

Algorithm 2 Generalized pattern search algorithm implemented in *ModestPy*

Require: Initial guess vector \mathbf{x}_0 with N parameters, initial step s_0 , max. no. of tries k_{max} to decrease step

```

 $s \leftarrow s_0$ 
 $k \leftarrow 0$ 
 $\mathbf{x} \leftarrow \mathbf{x}_0$ 
 $y \leftarrow f(\mathbf{x})$ 
while  $k < k_{max}$  do
   $y_n \leftarrow y$ 
  for  $n = 0$  to  $N$  do
     $\hat{\mathbf{x}}_n \leftarrow N$ -dim. vector with  $n$ -th element equal to 1
     $\mathbf{s}_n \leftarrow s\hat{\mathbf{x}}_n$ 
     $y_n^+ \leftarrow f(\mathbf{x} + \mathbf{s}_n)$ 
     $y_n^- \leftarrow f(\mathbf{x} - \mathbf{s}_n)$ 
    if  $y_n^+ < y_n^-$  then
       $y_n \leftarrow y_n^+$ 
       $\mathbf{x}_n \leftarrow \mathbf{x} + \mathbf{s}_n$ 
    end if
    if  $y_n^- < y_n$  then
       $y_n \leftarrow y_n^-$ 
       $\mathbf{x}_n \leftarrow \mathbf{x} - \mathbf{s}_n$ 
    end if
  end for
  if  $y_n < y$  then
     $y \leftarrow y_n$ 
     $\mathbf{x} \leftarrow \mathbf{x}_n$ 
     $k \leftarrow 0$ 
  else
     $s \leftarrow s/2$  {reduce step}
     $k \leftarrow k + 1$ 
  end if
end while
return  $\mathbf{x}$ 

```

```

inputs, # pandas DataFrame
known, # dictionary
estimate, # dictionary
measured, # pandas DataFrame
method=('GA', 'GPS'),
ga_opts={'maxiter': 5, 'tol': 1e-4},
gps_opts={'maxiter': 500, 'tol': 1e-6},
ftype='MSE'
)

estimates = session.estimate()
err, res = session.validate()

```

In addition to the estimation and validation results returned by the respective methods, the results are saved in the working directory together with plots of error and parameter trajectories.

3 Example

3.1 System

The functionality of the tool is presented on a case study in which it was used to calibrate a Modelica-based gray-box model of a single building zone to the results of a white-box model developed in EnergyPlus. The case study is a part of a larger project aimed at the development of an MPC framework that will be tested on the OU44 building in Odense, Denmark (Jradi et al., 2017). The developed EnergyPlus model is a downscaled 7-zone version of the OU44 building, but with the same HVAC system and internal heat gains schedules. The EnergyPlus model is used for MPC tests in the project.

The building is equipped with a balanced mechanical ventilation system. The air handling unit (AHU) contains two fans (for supply and exhaust air), a rotary wheel heat recovery system, and a heating coil. No cooling is provided. The fans are controlled to maintain a constant pressure in the duct system. Each zone is equipped with a hydronic radiator and a VAV box. The radiator valve and ventilation damper positions depend on the indoor temperature and CO₂ concentration, respectively. The temperature and CO₂ setpoint schedules are set through the BMS system.

Selected outputs of the white-box model are assumed to represent the measured data for the calibration of the gray-box model. The following zone-level outputs are used: indoor temperature T [°C], indoor CO₂ concentration CO_2 [ppm], radiator heat supply q_{rad} [W], ventilation airflow rate $verate$ [m³/s], and the number of occupants occ [–] (notation consistent with the gray-box model variables). The chosen outputs correspond with the available sensors in the OU44 building, so in the later phase of the project the same gray-box model can be used with the real data. In this study, however, the white-box results are used to exclude the effect of uncertain inputs on the results. In addition to the zone-level measurements, the weather data and assumed temperature and CO₂ setpoints are passed to the gray-box model as inputs.

3.2 Modelica Gray-box Model

The gray-box model was developed in Modelica and exported to an FMU using Dymola (Fig. 2). The thermal part of the model is based on the RC network approach and contains two capacitors, one for the indoor air and one for the internal thermal mass, and one resistor representing the external walls. The solar gains and radiator heat gains are modeled with single gain blocks. The radiator heat supply is controlled by a PI controller, depending on the temperature setpoint (input) and calculated indoor temperature. The occupancy contributes to both the indoor heat gains and CO₂ generation. The metabolic heat generation per person is modeled as a linear function of the indoor temperature. The CO₂ balance is modeled with a custom block containing a steady state mass balance equation. The ventilation heat gain is also modeled with a cus-

tom block containing a steady state heat balance equation. The rest of the model is based on the components from the Modelica Standard Library.

The model takes six inputs and has six outputs. The inputs include the solar radiation solrad [Wm^{-2}], outdoor temperature T_{out} [$^{\circ}\text{C}$], number of occupants occ , ventilation air temperature setpoint T_{vestp} [$^{\circ}\text{C}$], indoor CO_2 setpoint CO2stp [ppm], and indoor temperature setpoint T_{stp} [$^{\circ}\text{C}$]. The outputs are the indoor temperature T [$^{\circ}\text{C}$], indoor CO_2 CO2 [ppm], ventilation airflow rate verate [m^3s^{-1}], total ventilation airflow vetot [m^3], radiator heating rate qrad [W], and total radiator heating energy Qrad [J].

Seven of the model parameters are estimated: outdoor wall resistance RExt [m^2WK^{-1}], indoor wall resistance RInt [m^2WK^{-1}], infiltration air change rate Vinf [h^{-1}], maximum ventilation air change rate maxVent [h^{-1}], internal thermal mass imass [$\text{JK}^{-1}\text{m}^{-2}$], solar heat gain coefficient shgc [-], and indoor air thermal mass tmass [$\text{JK}^{-1}\text{m}^{-3}$]. All parameters excepts shgc are scaled by the respective surface areas (e.g. external wall surface area for RExt) or the indoor volume (tmass). It should be noted that the infiltration rate parameter Vinf affects only the CO_2 balance, while the thermal effect of infiltration is included in the resistance RExt .

3.3 Estimation Setup

Table 1 provides the lower and upper bounds in addition to the initial guesses of the seven parameters to be estimated in this study. The bounds were chosen considering standard physical and technical specifications of buildings.

Table 1. Initial guess, lower and upper bounds of the gray-box model parameters.

| Parameter | Initial guess | Low. bound | Up. bound |
|------------------|---------------|------------|-----------|
| shgc | 5 | 0.1 | 10 |
| tmass | 50 | 1 | 100 |
| imass | 50 | 1 | 100 |
| RExt | 5 | 0.1 | 10 |
| RInt | 5 | 0.1 | 10 |
| Vinf | 5 | 0.1 | 10 |
| maxVent | 5 | 0.1 | 10 |

The cost function is based on the total normalized mean-square error NMSE_{tot} , calculated according to Eq. (2). Four white-box model outputs are taken into account: T , CO_2 , verate , and qrad .

9 different method sequences were used to estimate the parameters: (1) GA, (2) GPS, (3) TNC, (4) SLSQP, (5) L-BFGS-B, (6) GA+GPS, (7) GA+TNC, (8) GA+SLSQP, (9) GA+L-BFGS-B.

The GA settings were consistent across the sequences (1) and (6)-(9). In addition a random number seed was used to make the results comparable. The maximum number of generations was set to 100. The initial population contained 40 individuals and was generated using the

Latin hypercube sampling. The maximum number of iterations in all the other methods was 500. The absolute solution tolerance was $1\text{e-}9$ (same for all methods).

All the non-default estimation parameters are specified in the following code:

```
ga_opts = {
    'maxiter': 100, 'tol': 1e-9,
    'lhs': True, 'pop_size': 40
}
gps_opts = {
    'maxiter': 500, 'tol': 1e-9
}
scipy_opts = {
    'solver': 'scipy_solver',
    'options': {
        'maxiter': 500, 'tol': 1e-9
    }
}
session = Estimation(
    wdir, fmu, inp, known, est, out,
    lp_frame=(0, 86400 * 3),
    vp=(86400 * 3, 86400 * 4),
    ga_opts=ga_opts,
    gps_opts=gps_opts,
    scipy_opts=scipy_opts,
    methods=met_seq,
    ic_param=ic_param,
    ftype='NMSE', seed=1
)
```

where scipy_solver is one from ['TNC', 'L-BFGS-B', 'SLSQP'], and met_seq is one from [('GA',), ('GPS',), ('SCIPY',), ('GA', 'GPS'), ('GA', 'SCIPY')].

The training period was 3 days long (January 1-3). The validation was performed on the day following the training period (January 4).

3.4 Results

All the 9 considered method sequences yielded different estimates, despite the same bounds and initial guesses (in GPS, SLSQP, L-BFGS-B, TNC) and the random number seed (in the GA-based sequences). Fig. 3 presents a stacked histogram of the obtained estimates. The estimates obtained by the GA-based sequences are close to the ones yielded by the GA alone. Differences are noticeable mainly in the case of shgc and maxVent . The remaining method sequences (non-GA-based) yielded estimates scattered across the parameter space.

The GA+L-BFGS-B and GA+GPS sequences yielded the lowest training errors, while GA+SLSQP was the most accurate in the validation, suggesting that the parameters are slightly overfitted in former cases (Table 2). GA+L-BFGS-B was the only GA-based sequence that performed worse in the validation than the GA alone. Nevertheless, all GA-based sequences yielded models with a similar accuracy, with training errors below 0.394 and validation errors below 0.379. The rest of the methods (GPS, TNC, L-

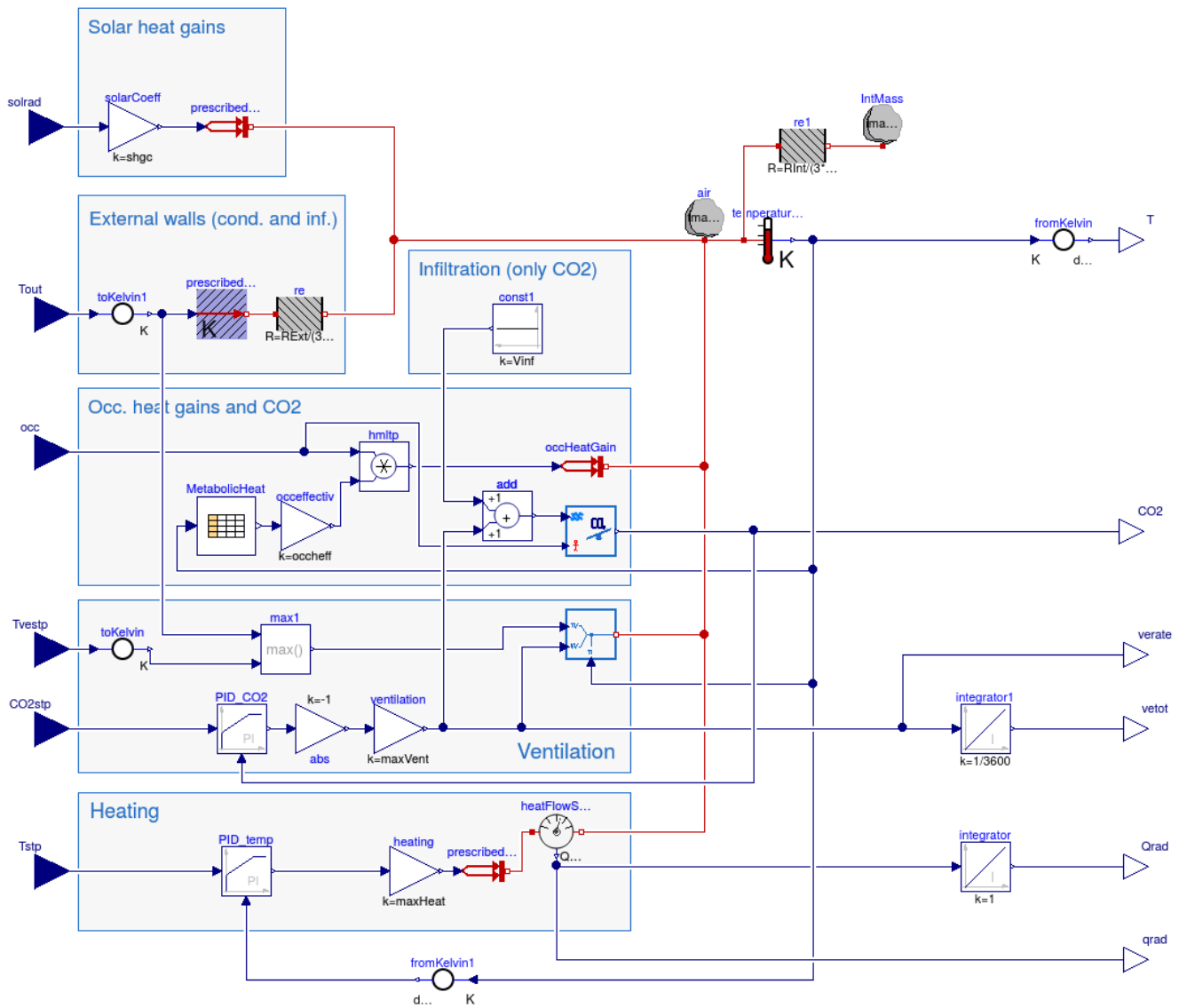


Figure 2. Gray-box zone model developed in Modelica (using Dymola).

BFGS-B, SLSQP) yielded significantly worse estimates, with validation errors above 3.4.

The computational time of the GA-based sequences was mostly dominated by GA (723 s). In the considered problem only GPS was slower than GA (986 s). All gradient-based methods included in the test were much faster, but stuck in local minima. From the time-to-accuracy point of view mixing GA with a gradient-based method seems the most efficient (801-934 s), while GA+GPS is the slowest combination (1319 s).

All computations were performed on a laptop equipped with an Intel Core i7-5600U processor (2.60GHz), 16 GB RAM, and a hard-disk drive (HDD). The disk type is relevant, because *ModestPy* was run in a verbose logging mode, each time saving a log file containing up to 85000 lines with detailed results from each iteration. All simulations were run on a single core – *ModestPy* does not offer parallelized algorithms yet.

Table 2. CPU time and total normalized mean square error ($NMSE_{tot}$) for validation and training, sorted in ascending order by validation $NMSE_{tot}$.

| Method | Training $NMSE_{tot}$ | Validation $NMSE_{tot}$ | CPU Time [s] |
|-------------|-----------------------|-------------------------|--------------|
| GA+SLSQP | 0.377 | 0.353 | 920 |
| GA+GPS | 0.351 | 0.371 | 1319 |
| GA+TNC | 0.393 | 0.372 | 801 |
| GA | 0.394 | 0.373 | 723 |
| GA+L-BFGS-B | 0.349 | 0.379 | 934 |
| GPS | 1.306 | 3.428 | 986 |
| TNC | 4.967 | 5.856 | 101 |
| L-BFGS-B | 4.929 | 6.808 | 38 |
| SLSQP | 5.040 | 6.920 | 12 |

The different estimates yielded by the respective methods can be due to inaccuracies in the numerically approximated gradients, and due to the non-convexity of the cost function. The non-convexity of the cost function used in this study can be analyzed in Fig. 4. The cost function evaluated on the line connecting estimates yielded by two different methods (GA and SLSQP) has few sections with positive derivatives and local minima. In addition the cost function is very steep in the vicinity of the GA solution, while relatively flat for $s < 0.7$. Although the shape of the function in other directions is not presented, Fig. 4 highlights the need for a good initial guess in the case of gradient-based methods if the cost function is non-convex.

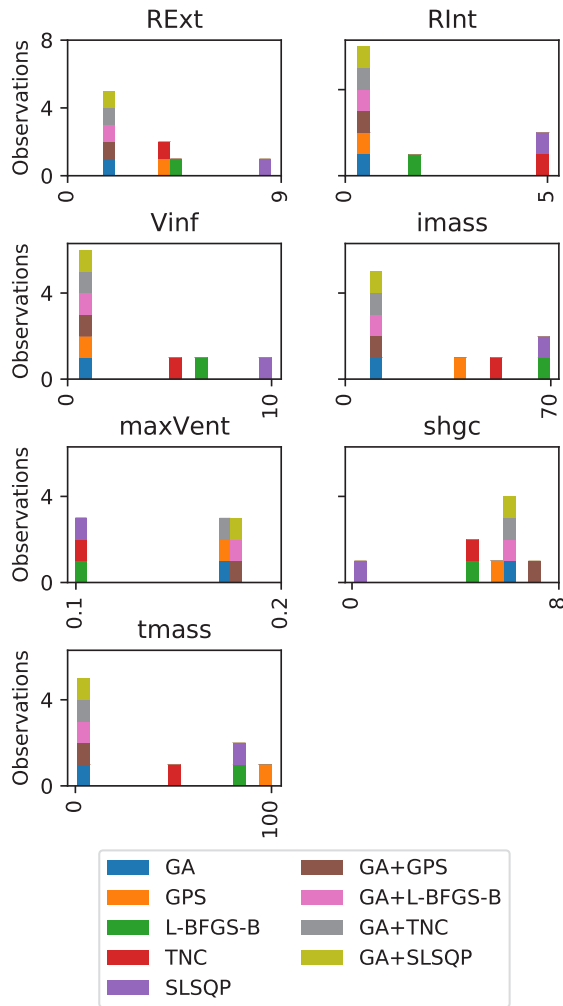


Figure 3. Histogram of estimates yielded by the 9 method sequences.

In the case of non-convex problems, it is sometimes useful to analyze the visualization of the parameter search in GA (Fig. 5). This figure is by default generated by *ModestPy* whenever GA is used in the method sequence. Each dot in the figure represents a solution produced by a single individual. The colors represent the error (NMSE_{tot}), with the brightness decreasing with decreasing error. The GA starts with the inaccurate population spread over the entire

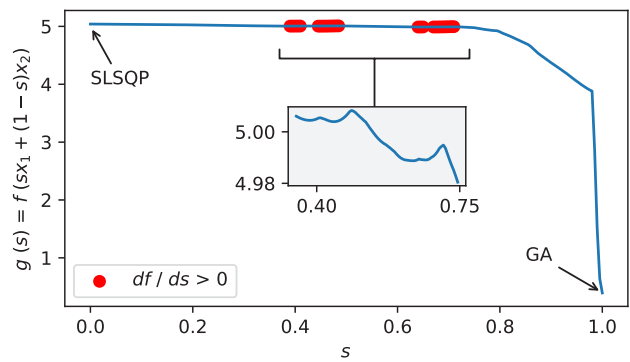


Figure 4. Cost function evaluated on the training data based on linear combinations of parameters yielded by GA (x_1) and SLSQP (x_2). Sections with positive derivatives with respect to s marked in red.

search domain (yellow color marks NMSE_{tot} above 5.0). Over time, the solution quality improves to NMSE_{tot} below 1.0 (purple color). For some parameters (most notably maxVent and Vinf) the value found early in the evolution does not change much throughout the rest of the evolution. In other cases rapid jumps in parameters with only a minor improvement in the accuracy are observed (e.g. imass). Based on this visualization, the user is able to assess whether a desired balance of the exploration and local search was achieved.

Fig. 6 depicts root-mean-square errors (RMSE) for each output variable, calculated for the validation period. RMSE was not used in the cost function, because it is non-differentiable at 0 which is problematic for gradient-based methods, but the metric helps interpreting results. RMSE represents the standard deviation between the predicted and true values. The analysis of the errors reveals that the problems encountered by the non-GA-based sequences were due to both the thermal and CO₂ parts of the model. The best performing models achieved RMSE below 50 ppm for CO₂, 0.4 °C for T, 600 W for qrad, and 10 m³h⁻¹ for verate. The RMSE differences between the best and worst performers are around 180 ppm for CO₂, 2 °C for T, 1600 W for qrad, and 20 m³h⁻¹ for verate. Interestingly, unlike the gradient-based methods, GPS calibrated well the parameters affecting CO₂ and verate, even though it started from the same initial guess.

The validation results for the four output variables explain the large errors yielded by the non-GA-based sequences. The temperature in those cases slowly reacts to the applied heat loads (Fig. 7), which is due to the overestimated thermal mass (tmass, imass) of the building (Fig. 3).

In the case of CO₂ (CO₂) and ventilation rate (verate) the inaccuracy of the gradient-based methods is due to the overestimated infiltration rate Vinf (Fig. 3), which reduced the indoor CO₂, preventing it from reaching the setpoint of 800 ppm and triggering the ventilation. Hence, the ventilation in those cases remained turned off through-

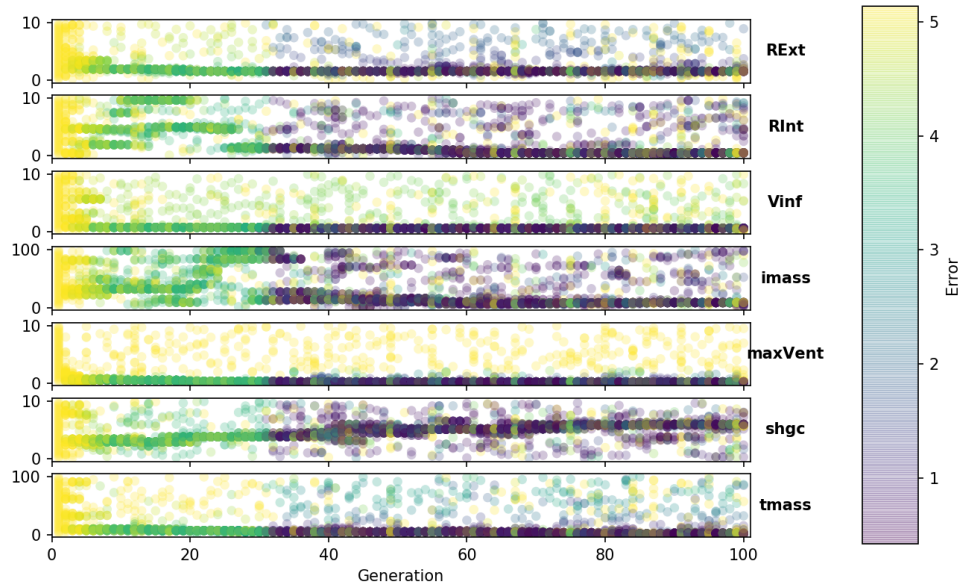


Figure 5. Parameter evolution in the genetic algorithm – color represents the training error (darker more accurate).

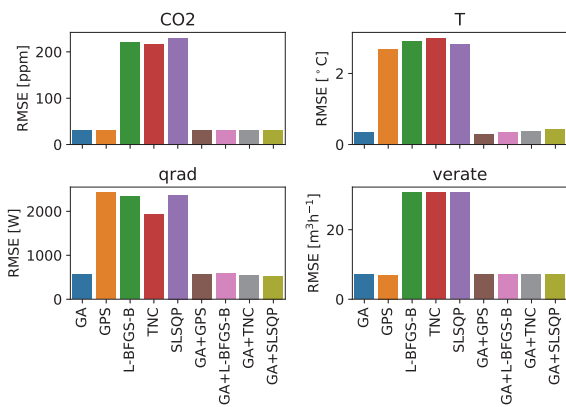


Figure 6. Validation root-mean-square error (RMSE) per output variable.

out the entire validation period (*verate* equal to 0).

Similarly, the models with the inaccurate indoor temperature yielded inaccurate radiator heating power *qrad* (Fig. 8). Only the GA-based method sequences were able to achieve a good accuracy with respect to *qrad*. In all cases, however, there is a mismatch in the initial 8 hours of the validation period due to the wrong initial condition for *qrad*. If the grey-box model is to be used for short-term predictions, the initial value should be based on the measurements, as in the case of indoor temperature and CO₂.

4 Discussion

The results are not generalizable to other models, estimation settings (initial guesses, number of iterations, parame-

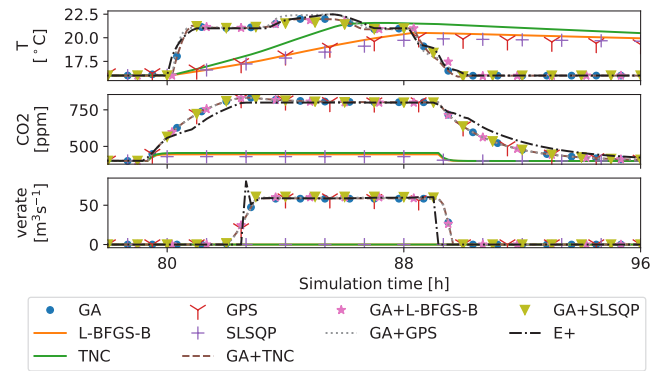


Figure 7. Validation results: temperature (T), CO₂ (CO2), ventilation airflow rate (*verate*).

ter bounds etc.), or even training data. The specific methods can perform differently on different problems. The gradient-based methods heavily rely on the quality of the initial guess, while the GA results depend on the random seed. Both, the initial guess and the random seed were fixed in this study. However, the results highlight the importance of such inter-method comparisons, especially in the cases where the parameter estimation is a non-convex problem. In many practical applications it may be difficult to assess whether or not the problem is non-convex, or what should be the initial guess for parameters.

The GA results presented in this paper are surprisingly good in terms of the accuracy and computational time. Typically GA requires much more time to converge than gradient-based methods. However, since GA is stochastic in nature, the results could be different if the experiment was repeated without a fixed random seed. The computa-

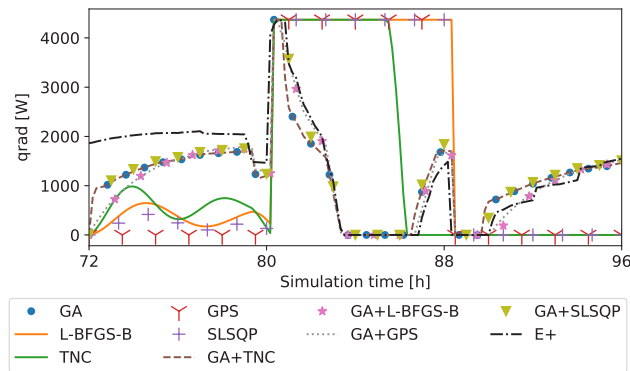


Figure 8. Validation results: radiator heating rate (q_{rad}).

tional times reported in this study are based on single trials and therefore may be slightly biased.

The authors advise to always couple GA with another gradient-based method and rerun the estimation several times if there are no strict time constraints, possibly with various GA settings. *ModestPy* allows for setting up the number of estimation runs with an optionally moving training period during the instantiation of the *Estimation* class. An alternative approach could be to test the gradient-based methods with multiple random initial guesses.

It should be noted, that the gradient-based methods (SLSQP, L-BFGS-B, TNC) would be much faster if the gradient of the cost function with respect to the estimated parameters was known. In a general case, this gradient is not known and has to be evaluated numerically, as in this study. However, the FMI standard allows to provide directional derivatives (optional feature), and some tools support it. If the computational time is an issue, other tools that are able to perform algebraic differentiation should be used, e.g. JModelica.org (Åkesson et al., 2009).

5 Conclusions

Automated parameter estimation is crucial in many industrial applications, including MPC and other cyber-physical systems. The FMI standard provides an attractive simulation interface that allows for using models outside their dedicated environments, and developing model-agnostic tools. This paper introduces a new Python tool for parameter estimation in FMI-compliant models, called *ModestPy*. The tool supports several optimization methods, both gradient-free and gradient-based (numerically approximated), that can be queued in user-defined sequences. *ModestPy* enables easily testing multiple methods on a given model and find the most suitable approach and estimation settings.

The tool was tested on a case study in which it was used to estimate parameters of a non-linear multi-output model of a building zone. The results showed that the local optimization methods (GPS, L-BFGS-B, TNC, SLSQP) were unable to calibrate the model, marking that the the initial guess for parameters was poor. The addition of GA

as the initial global search method significantly improved the model accuracy. The results also validated the two in-house algorithms (GA and GPS).

It should be noted, that the initial global search would not be needed if the approximate initial values of parameters were known. In such a case the gradient-based methods would easily outperform GA.

The current functionality of the tool is already sufficient for a general use. It is used by the authors for calibrating gray-box models of buildings and HVAC systems for the use in MPC.

However, the development work continues and there are plans to include the following functionality:

- a simple graphical user interface to attract users less experienced in the Python programming language,
- support for on-line estimation methods (e.g. Kalman filter),
- support for multi-period stochastic gradient descent training,
- support for parallel processing methods.

6 Acknowledgments

The authors thank David Blum for comments that helped to improve the manuscript and the development of *ModestPy*.

This work was supported by the Innovation Fund Denmark for the project COORDICY (4106-00003B).

This research was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy, under Contract No. DE-AC02-05CH11231.

This work emerged from the IBPSA Project 1, an international project conducted under the umbrella of the International Building Performance Simulation Association (IBPSA). Project 1 will develop and demonstrate a BIM/GIS and Modelica Framework for building and community energy system design and operation.

References

- Johan Åkesson, Magnus Gäfvert, and Hubertus Tummescheit. JModelica—an open source platform for optimization of Modelica models. In *Proceedings of MATHMOD 2009 - 6th Vienna International Conference on Mathematical Modelling*, Vienna, Austria, February 2009. TU Wien.
- Christian Andersson, Sofia Gedda, Johan Åkesson, and Stefan Diehl. Derivative-free parameter optimization of functional mock-up units. In *Proceedings of the 9th International Modelica Conference - Munich, Germany, September 3, 2012*. Modelica Association, 2012.
- Christian Andersson, Johan Åkesson, and Claus Führer. *PyFMI: A Python Package for Simulation of Coupled Dynamic Models with the Functional Mock-up Interface*, volume LUTFNA-5008-2016 of *Technical Report in Mathematical*

- Sciences. Centre for Mathematical Sciences, Lund University, 2016.
- Krzysztof Arendt. ModestPy: Parameter Estimation in FMI-compliant Models, 2017. URL <https://github.com/sdu-cfei/modest-py>. [Online; accessed April 25, 2018].
- Javier Bonilla, Jose Antonio Carballo, Lidia Roca, and Manuel Berenguel. Development of an open source multi-platform software tool for parameter estimation studies in fmi models. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, number 132, pages 683–692. Linköping University Electronic Press, Linköpings universitet, 2017.
- Marco Bonvini, Michael Wetter, and Michael D. Sohn. An FMI-based Framework for State and Parameter Estimation. In *Modelica Conference 2014*, 2014.
- Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208, September 1995. ISSN 1064-8275. doi:10.1137/0916069.
- Roel De Coninck and Lieve Helsen. Practical implementation and evaluation of model predictive control for an office building in brussels. *Energy and Buildings*, 111:290 – 298, 2016. ISSN 0378-7788. doi:<https://doi.org/10.1016/j.enbuild.2015.11.014>.
- Roel De Coninck, Fredrik Magnusson, Johan Åkesson, and Lieve Helsen. Toolbox for development and validation of grey-box building models for forecasting and control. *Journal of Building Performance Simulation*, 9(3):288–303, 2016. doi:10.1080/19401493.2015.1046933.
- Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001. URL <http://www.scipy.org/>. [Online; accessed April 25, 2018].
- Muhyiddine Jradi, Fisayo Caleb Sangogboye, Claudio Giovanni Mattered, Mikkel Baun Kjærgaard, Christian Veje, and Bo Nørregaard Jørgensen. A world class energy efficient university building by danish 2020 standards. *Energy Procedia*, 132:21 – 26, 2017. ISSN 1876-6102. doi:<https://doi.org/10.1016/j.egypro.2017.09.625>. 11th Nordic Symposium on Building Physics, NSB2017, 11-14 June 2017, Trondheim, Norway.
- Rüdiger Kampfmann, Danny Mösch, and Nils Menager. Parameter Estimation based on FMI. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, number 132, pages 313–319. Linköping University Electronic Press, Linköpings Universitet, 2017.
- Dieter Kraft. *A Software Package for Sequential Quadratic Programming*. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln: Forschungsbericht. Wiss. Berichtswesen d. DFVLR, 1988.
- Stephen G. Nash. Newton-Type Minimization Via the Lanczos Method. *SIAM Journal on Numerical Analysis*, 21(4):770–788, 1984. ISSN 00361429.
- Peter Rockett and Elizabeth Abigail Hathway. Model-predictive control for non-domestic buildings: a critical review and prospects. *Building Research & Information*, 45(5):556–571, 2017. doi:10.1080/09613218.2016.1139885.
- Fisayo Caleb Sangogboye, Krzysztof Arendt, Ashok Singh, Christian T. Veje, Mikkel Baun Kjærgaard, and Bo Nørregaard Jørgensen. Performance comparison of occupancy count estimation and prediction with common versus dedicated sensors for building model predictive control. *Building Simulation*, 10(6):829–843, Dec 2017. ISSN 1996-8744. doi:10.1007/s12273-017-0397-5.
- Luigi Vanfretti, Maxime Baudette, Achour Amazouz, Tetiana Bogodorova, Tin Rabuzin, Jan Lavenius, and Francisco José Gómez-López. Rapid: A modular and extensible toolbox for parameter estimation of modelica and fmi compliant models. *SoftwareX*, 5:144 – 149, 2016. ISSN 2352-7110. doi:<https://doi.org/10.1016/j.softx.2016.07.004>.
- Michael Wetter. Genopt - a generic optimization program. In Roberto Lamberts, Cezar O. R. Negrão, and Jan Hensen, editors, *Proc. of the 7th IBPSA Conference*, volume I, pages 601–608, Rio de Janeiro, 2001. URL http://www.ibpsa.org/proceedings/BS2001/BS01_0601_608.pdf.

A Safe Regression Test Selection Technique for Modelica

Niklas Fors¹ Jon Sten² Markus Olsson² Filip Stenström²

¹Department of Computer Science, Lund University, Sweden, niklas.fors@cs.lth.se

²Modelon AB, Sweden, jon.sten@gmail.com, {[filip.stenstrom](mailto:filip.stenstrom@modelon.com)|[markus.olsson](mailto:markus.olsson@modelon.com)}@modelon.com

Abstract

Running regression tests for Modelica models usually takes a long time. This paper presents a safe regression test selection technique for Modelica based on static analysis. The technique tracks dependencies between classes to compute which tests that need to be run given a change. The dependency rules have been verified using mutation testing. The technique has been evaluated on the Modelica Standard Library and another library with promising results.

Keywords: regression test selection, mutation testing

1 Introduction

Regression testing is an important activity when developing software today, preventing software changes from introducing bugs that break previous functionality. Regression tests allow the developer to run tests while developing for spotting errors early on. However, running all tests may take a long time, which makes the developer less likely to run the tests very often. *Regression test selection* tries to solve this problem by running a subset of all tests. There are different test selection techniques (Rothermel and Harrold, 1996; Yoo and Harman, 2012), some of them are *safe*, meaning that all tests that may be affected by the change are selected. Safe techniques are usually approximate and include tests that are not affected by the change. These techniques may also have a higher runtime for computing the selection, which should be lower than the time saved using the selection compared to running all tests.

This paper presents a safe regression test selection technique for the modeling language Modelica (2018), which is based on *extraction-based test selection* introduced by Öqvist et al. (2016) for Java. Tests in Modelica usually require relatively long compilation and simulation time, which makes test selection especially suitable for Modelica. For example, we have evaluated the technique on the Modelica Standard Library (MSL), where running all tests takes about 2-3 hours. If a random class is changed in MSL, then on average 95.5% time can be saved by running the selected tests compared to running all tests. The runtime for the test selection algorithm is only 0.14% of running all tests.

Our technique analyzes the source code and finds dependencies between classes, which form a dependency graph. The test selection algorithm takes a set of changed

classes and gives back the tests that depend on the changed classes, according to the dependency graph.

The contributions of this paper are the following:

- Dependency rules that describe when a class depends on another class (Section 3). These rules include name bindings, nested classes, `redeclare` clauses, and implicitly called functions such as `equalityConstraint`.
- Empirical verification of the dependency rules using mutation testing (Section 4). The mutation testing was performed on MSL, where each class was mutated to detect which tests that actually depend on the mutated class. The actual dependencies were then compared with the output of the test selection algorithm.
- Evaluation of how much time is saved using test selection on two libraries, the Modelica Standard Library and the Heat Exchanger Library (Section 5).
- An open source test suite for dependency analysis algorithms for Modelica¹.

The work in this paper has been carried out as a master's thesis project by Olsson and Stenström (2018) at Modelon, and is a continuation of the master's thesis project by Hedblom and Rundquist (2017)² with higher precision.

2 Safe Test Selection

The test selection algorithm takes a set of changed classes and returns a set of test cases that need to be run. Consider the set of all classes C in a system, and the test cases T that is a subset of C , and which are considered as entry points. For a change in a subset of C , the test selection algorithm gives back a subset $T_{ts} \subseteq T$ that need to be run due to the test results may have changed. The algorithm is *safe*, meaning that the test cases T_f that actually fail due to the changes is a subset of T_{ts} (hence, $T_f \subseteq T_{ts} \subseteq T \subseteq C$).

The test selection is implemented by tracking dependencies between classes using static analysis. A class X *depends* on another class Y if a change in Y may affect the meaning of X . The dependencies form a dependency graph, which is used by the test selection algorithm to

¹See <https://github.com/modelon/MCDTS>

²Hedblom and Rundquist were also supervised by Niklas Fors.

```

model A
...
end A;

model B
A a;
end B;

model T1
A a;
end T1;

model T2
B b;
end T2;

```

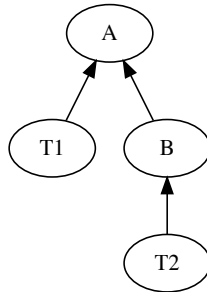


Figure 1. Two test cases T1 and T2 that use the models A and B, respectively. The right part of the figure shows the dependencies between the models.

find dependent test cases given a set of changes. We use the term *class* in accordance with the Modelica specification, which includes *specialized classes* such as `model`, `record`, `type`, `function`, etc.

For example, consider the Modelica source code in Figure 1. In this example, we have two test cases T1 and T2 that use the models A and B, respectively. The figure also shows the dependency graph between the classes. If a change is made in model A, then the dependency graph tells us that test case T1 and T2 need to be rerun. However, if a change is made in model B instead, then only T2 needs to be rerun. As illustrated by the example, the dependencies are transitive, for instance, test case T2 depends transitively on A.

2.1 Detecting Changes

Detecting changes can be done in different ways. One approach is to detect file changes, for instance, using version control systems, and then map files to classes. For example, if a file has changed, then all classes in that file can be considered as changed. This is the approach we currently use and which was implemented during the previous master’s thesis. It was chosen because it was the easiest approach to implement. The effect of the approach depends on how the Modelica code is organized. For example, MSL usually has quite many classes per file (on average 30 per file) and there are other libraries that have fewer classes per file. A more fine-grained approach would analyze what part of the file that has changed and which classes that corresponds to.

2.2 External Code

Modelica supports interfacing with other languages, such as C and FORTRAN. This can be challenging as it is very hard to calculate the dependencies within the external code. It is also common that the external code, based on input from the Modelica code, will read and access other resources, e.g. files or network resources.

In the current implementation, changes to non-Modelica files will mark all Modelica classes with external dependencies as changed. This, in turn, will force a rerun of all test cases which directly or indirectly depend on external code. This is not ideal as, to the user, seemingly harmless changes may mark some test cases for rerun. Possible improvement is to allow the user to provide a list of files to monitor or not to monitor for changes. For example, by allowing the user to list a set of files which won’t cause rerun of tests, or provide a list of files which will force a rerun of tests.

3 Dependency Rules

The following dependency rules are used for building the dependency graph between classes:

Rule 1. A class has a dependency on an accessed class, including all parts of the qualified name. This includes component declarations, extending clauses, function calls, import statements, functions in annotations (derivative, inverse) and overloaded operators.

Rule 2. A class has a dependency on its enclosing class.

Rule 3. A class that contains a redeclaration depends on all super classes and enclosed classes of the replacing class (and all their enclosed classes and super classes recursively).

Rule 4. A class has a dependency on implicitly called classes. This includes a record or type enclosing a function named `equalityConstraint`, and a class extending the class `ExternalObject` has dependency on enclosed function `destructor`.

3.1 Motivation

The rules will now be motivated with examples.

Rule 1 was illustrated in Section 2, namely that a class depends on classes it references. Additionally, the rule also handles qualified accesses. For example, for the access `A.B.C`, the rule creates dependencies to `A`, `A.B` and `A.B.C`. This is needed because changes in `A` or `A.B` may change what `C` refers to. For instance, class `C` may be declared in a supertype of `A.B`, and changing the supertype of `A.B` may then change what `C` refers to.

Rule 2 is illustrated in Figure 2. If model B is changed to extend `A2` instead of `A1`, then the type of `m` in `B.C` is changed from `A1.M` to `A2.M`. Thus, Rule 2 is needed to create a dependency from `B.C` to `B` to handle when B changes.

Rule 3 is illustrated in Figure 3. The dependency analysis needs to be careful with redeclare clauses. In the figure, the class `C` needs a dependency to all nested classes in `A2` due to the redeclare modifier in component declaration for `b`. This because the replaceable package `P` in package `B` is redeclared to `A2` in the context of component `b` in

```

package A1
  model M
  end M;
end A1;

package A2
  model M
  end M;
end A2;

package B
  extends A1;

  model C
    M m;
  end C;
end B;

```

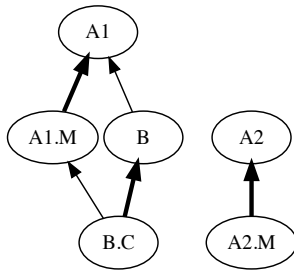


Figure 2. Example for Rule 2. Dependencies stipulated by Rule 2 are illustrated with thicker edges. The rule is needed to handle if B changes its extends clause to A2, which changes the reference in class B.C.

model C. This broad addition of dependencies is needed in order to capture the actual use of A2.f in package B when in context of model C. Alternatively, a complex and time consuming analysis of the usages of the package P in context of model C could be done. Note that the rule is recursive, meaning that if A2 would have a nested class NC1 which in turn would have another nested class NC2, then C would depend on both NC1 and NC2.

Rule 3 also handles classes prefixed with `redeclare`, which is illustrated in Figure 4. In this example, we want a dependency from B to B.f, since B redeclares the function f that is declared in A.

Rule 4 is needed due to specific dependencies that the language specification dictates based on class context. See the test suite for examples of Rule 4.

3.2 Implementation

We implemented the dependency analysis based on the rules in this section in the OPTIMICA Compiler Toolkit³, which is based on the JModelica.org compiler (Åkesson et al., 2010a).

4 Verification

We want the test selection algorithm to be safe, but it is hard to know if the dependency rules (Section 3) cover all cases since Modelica is a complicated language. We have verified the dependency rules empirically by using tests from the previous master’s thesis project (Hedblom and Rundquist, 2017), coming up with new tests manually and performed verification based on mutation testing.

Mutation testing is a technique for evaluating how efficient a test suite is (DeMillo et al., 1978). This is done by automatically introducing small faults in the program

```

package A1
  function f
  end f;
end A1;

package A2
  function f
  end f;
end A2;

package B
  replaceable
  package P = A1;
  Real x = P.f();
end B;

model C
  B b(redeclare
  package P = A2);
end C;

```

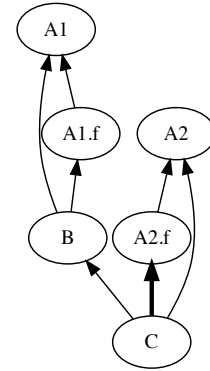


Figure 3. Example for Rule 3, which creates a dependency from C to A2.f. This dependency is needed to handle changes in A2.f.

```

model A
  replaceable
  function f
  end f;
  Real x = f();
end A;

model B
  extends A;

  redeclare
  function f
  end f;
end B;

model C
  B b;
end C;

```

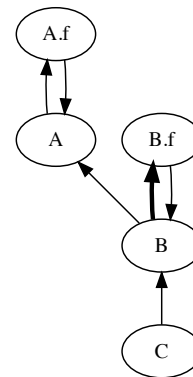


Figure 4. Another example for Rule 3 that illustrates classes with the redeclare prefix, which creates a dependency from B to B.f.

³<http://www.modelon.com/products/modelon-creator-suite/optimica-compiler-toolkit/>

```

model A;
  Real x;
equation
  x = 7;
end A;

model B
  A a;
  Real y;
equation
  y = a.x + 3;
end B;

model T
  B b;
end T;

fclass T;
  Real b.a.x;
  Real b.y;
equation
  b.a.x = 7;
  b.y = b.a.x + 3;
end T;

```

(a) Source system

(b) Flat class

Figure 5. Model T is selected and the compiler instantiates it to a flat class.

and then checking if these faults are detected by any test case. If a fault is not detected, then a new test case can be added that covers the fault. A change may be, for example, switching the branches in an if statement or replacing arithmetic operators, for instance, replacing a minus (−) with a plus (+). However, we use this technique in a little different way since we are interested in verifying the test selection algorithm, and thus are interested in finding out the actual dependencies between classes. We do this by introducing changes in a class and then computing which tests that are actually affected by the changed class. For the affected tests, there is an actual dependency from the test to the changed class, and we want this dependency to be in the dependency graph. If this is not the case, then the algorithm contains a bug or a dependency rule is missing.

We only change one class at a time and then check which tests are actually affected by the change. We detect that a test is affected by the change by computing and comparing the string representation of the flat (unoptimized) class for the test. This flat class contains all the variables, equations, function and other parts of the model which is needed in order to solve the equation system. If the flat class has changed in any way, we consider that the test is affected by the change. Note that running the test may actually give the same results as before since a test involves comparing simulation results. However, we are only interested in the dependencies to the changed class from the test class, and if the flat test class changes in any way, then there is a dependency (direct or indirect).

When a test is selected to run, the compiler creates a flat class representing the equation system for the test, which is then optimized and later solved using a numerical solver. The flat class is illustrated in Figure 5, where the test T has been selected and instantiated by the compiler. As can be seen, the object-oriented and hierarchical structure are removed in the flat class (but not optimized

Table 1. Mutation testing on MSL. The unique column is how many classes the mutation was unique for. Mutations is the total number of mutations carried out. The total number of classes in MSL is 5946.

| Mutation | Mutated classes | Unique | Mutations |
|----------|-----------------|--------|-----------|
| LocalVar | 1772 (29.5%) | 485 | 1772 |
| Arit | 2026 (33.8%) | 16 | 9647 |
| Lit | 3306 (52.3%) | 805 | 25563 |
| Bool | 913 (15.2%) | 17 | 3028 |
| Redecl | 68 (1.1%) | 17 | 68 |
| Comment | 1650 (27.3%) | 205 | 10142 |

in the figure). For this class, we could change the value of x in model A to 5, and the flat class would change, and we can then infer that there is an indirect dependency from T to A. To verify that the test selection algorithm works, we check that this dependency is in the dependency graph. If we instead would change the value of x to $4 + 3$, then the flat class would still change, even if the meaning would remain (since $4 + 3 = 7$). This is as earlier described not an issue, since we are interested in computing actual dependencies between classes.

4.1 Mutation Testing on MSL

We performed mutation testing on MSL with the following types of mutations.

LocalVar. Add local variable to function.

Arit. Switching arithmetic operands, e.g., $1 - 2 \Rightarrow 2 - 1$.

Lit. Changing literals, e.g., $2 \Rightarrow 3$.

Bool. Changing boolean operators, e.g., $a < b \Rightarrow a > b$.

Redecl. Redeclaring a replaceable function.

Comment. Changing string comments.

Changing string comments will not change the meaning of the program, but they are carried over to the flat class, which is what we are interested in.

The mutation testing was performed for one class in MSL at the time and one mutation type at the time. However, several mutations of the same type could be applied for one class, for example, by changing several literals. The results are shown in Table 1. As can be seen in the table, changing literals was the most applicable mutation type and could be applied for more than half of the classes in MSL. Performing one mutation test took around 18 minutes, since it requires all tests to be individually instantiated to a flat class. We performed the mutation tests on a cluster of Jenkins machines, and it would take 280 days if they were carried out in a sequence (non-parallel).

From the result of mutation testing, we needed to generalize Rule 3, discovered Rule 4, found derivative/inverse in annotations, and found six bugs in the implementation. A test case was added to the test suite for dependency analysis algorithms for each fault found.

4.2 Threat to Validity

Note that we have only verified the test selection algorithm empirically and not formally. Thus, we cannot know with certainty that the rules and implementation are safe. A complete formal verification would require a formal model of Modelica, a language that is very complicated. It would also be possible to use a simplified formal model that does not cover the complete language, but some aspects of it. However, using a simplified formal model would probably miss, for example, equality constraints (Rule 4) because they are not a central part of the language, which the mutation testing did find. Also, the mutation testing was performed on MSL, which might not use all language features. Thus, it would be desirable with more mutation testing on other libraries to make the verification more complete.

4.3 Partial Dependency Graph

We also used mutation testing to compute a partial dependency graph between classes in MSL. For each mutated class, we get actual dependencies to the mutated class from all test classes that changed because of the mutation. Combining all actual dependencies from all mutations yield a partial dependency graph from test classes to classes. This partial dependency graph can be used to partly verify test selection algorithms. Thus, in addition to the manually created test cases for dependency analysis algorithms, the open test suite also contains the partial dependency graph from the mutation testing as an XML file.

4.4 Instrumenting the Compiler

We have used mutation testing to compute actual dependencies between test classes and classes. Another way would be to instrument the compiler to compute all actual dependencies when instantiating a test class, which would require less resources. The reason why we chose mutation testing is because we believe that instrumenting the compiler might contain the same bugs as the test selection implementation. Mutation testing is more like black-box testing in this context. However, it would be useful to complement the mutation testing with compiler instrumentation to improve the verification, which is something we would like to do.

4.5 Previous Technique Unsafe

During the mutation testing we found that the implementation by Hedblom and Rundquist (H&R) was not safe since it ignored classes with the `redeclare` prefixes. We also found another bug in their implementation that included too many dependencies. We fixed these two issues in the evaluation (Section 5) for H&R's technique when comparing it to our technique.

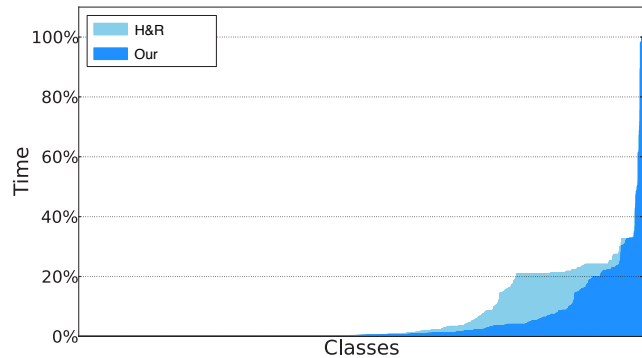


Figure 6. Tests runtime for each class changed in MSL. The number of classes is 5946, of which 366 are tests.

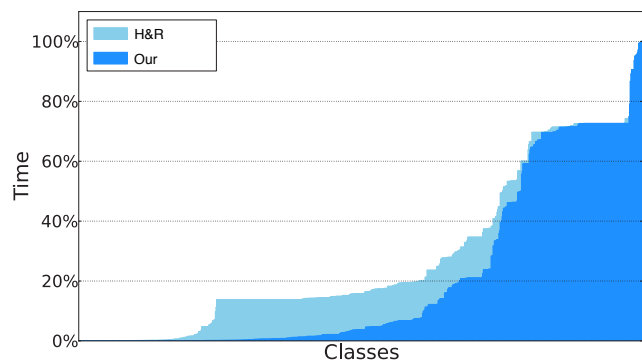


Figure 7. Tests runtime for each class changed in HXL. The number of classes is 871, of which 227 are tests.

5 Evaluation

We have evaluated the time saved using test selection compared to running all tests on two libraries, the Modelica Standard Library (MSL) and the Heat Exchanger Library (HXL) by Modelon⁴. The time saved is compared to the test selection defined by Hedblom and Rundquist (2017), which will be denoted H&R. Their technique is more coarse-grained than ours and operates in an earlier step in the compilation process, leading to more tests selected, but with a lower analysis running time (see Section 6 for comparison).

We computed the test selection for each class in the library and then the runtime for the selected tests, which includes compilation and simulation time. The runtime for the selected tests was computed as a percentage of the runtime for all tests. The results are shown in Figures 6-7, where classes are sorted by tests runtime in ascending order. For both techniques, the average savings is above 90% for MSL and above 70% for HXL. The average savings, the average test runtime, the median test runtime and the analysis time are shown in Tables 2-3 under *Class*. The complement of the average savings in the tables correspond to the blue areas in the graphs.

We also computed the test selection for each file in MSL and HXL, where all classes in a file were consid-

⁴<https://www.modelon.com/library/heat-exchanger-library/>

Table 2. Performance results for MSL. All values are in percentage of the time it takes to run all tests.

| | Class | | File | |
|-------------------|-------|-------|-------|-------|
| | Our | H&R | Our | H&R |
| Average savings | 95.5% | 93.1% | 88.9% | 87.9% |
| Average test time | 4.3% | 6.9% | 11.0% | 12.1% |
| Median test time | 0.22% | 0.33% | 1.19% | 1.7% |
| Analysis time | 0.14% | 0.04% | 0.14% | 0.04% |

Table 3. Performance results for HXL. All values are in percentage of the time it takes to run all tests.

| | Class | | File | |
|-------------------|-------|-------|-------|-------|
| | Our | H&R | Our | H&R |
| Average savings | 78.9% | 72.0% | 80.5% | 74.8% |
| Average test time | 20.9% | 27.9% | 19.4% | 25.1% |
| Median test time | 3.61% | 15.7% | 3.77% | 15.7% |
| Analysis time | 0.12% | 0.10% | 0.12% | 0.10% |

ered changed. The results can be seen in the tables under *File*. As expected, the time saved for files is less than for classes.

As we can see, the new test selection technique has higher precision than H&R’s technique leading to improved savings. However, our dependency analysis is a bit more advanced and the implementation is 201 source lines of code⁵ (SLOC) specified in JastAdd (Hedin and Magnusson, 2003), whereas H&R’s implementation is 134 SLOC.

5.1 MSL Commit History

We have also used the MSL commit history to get more realistic sets of changed files. Thus, we use each commit as a set of changed files and compute the time saved for that set. The results are shown in Figure 8, where commits are sorted by tests runtime in ascending order. The average saving is 68.9% for our technique and 57.0% for the old technique.

6 Related Work

As described earlier, the dependency rules presented in this paper is a continuation of the master’s thesis by Hedblom and Rundquist (2017), but with higher precision. H&R implemented their test selection algorithm in an earlier step in the compilation process⁶ with less static information available about name bindings etc., leading to more coarse-grained rules. One major difference is that the previous implementation could not resolve all parts of a qualified access like `a.b.c`. This lead to a dependency rule for class accesses where an access to a class `A` created dependencies to all nested classes enclosed by `A`

⁵Measured with *cloc*, see <https://github.com/AIDanial/cloc>

⁶H&R implemented their algorithm in the *source tree*, whereas our algorithm is implemented in *instance tree*, according to the different compilation steps defined by Åkesson et al. (2010b).

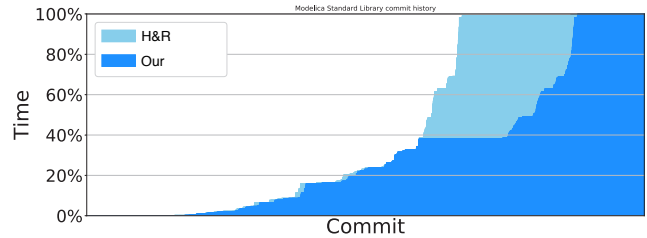


Figure 8. Test runtime for MSL commit history.

(recursively). Another difference is that H&R’s rules are implementation-specific, where the rules use the term *resolvable* to mean names that are resolved in that compilation step in the OPTIMICA compiler. In contrast, our rules are defined in terms of the language and are not dependent on the implementation. However, it would be possible to generalize the H&R’s rules to be implementation-neutral.

There is previous work on safe test selection for other languages (Chen et al., 1994; Rothermel and Harrold, 1997), such as Java (Öqvist et al., 2016; Gligoric et al., 2015), where both dynamic- and static analyses have been investigated. Our technique is based on *extraction-based test selection* by Öqvist et al. (2016), who applied it for Java. This kind of technique uses only static analysis and is more coarse-grained (for instance, dependencies between files or classes) than other techniques. Modelica is quite different from Java, leading to other dependency rules, for example, covering the `redeclare` mechanism. One interesting property from a safe test selection perspective is that compilation and simulation usually takes rather long time for Modelica models, especially when comparing to compilation and test time for Java. This makes test selection especially useful for Modelica. Also, since Modelica compilers generate flat equation systems for test classes, you cannot use dynamic analysis, but only static analysis.

7 Conclusions

We have in this paper presented a regression test selection technique using static analysis for Modelica with very promising results. In the evaluation, we found that changing a class in MSL and only running the tests selected by the algorithm saved on average 95.5% tests runtime compared to running all tests. Using MSL commit history as basis for changed files, then the average saving is 68.9%. Since Modelica is a complicated language, we performed mutation testing on MSL to verify that our dependency rules are correct and safe. However, with mutation testing, we cannot prove that the rules or the implementation of the rules are complete.

In the future, we would like to do more mutation testing on other libraries than MSL, and also add more mutation types. It would also be interesting to update the dependency graph incrementally. The current implementation computes the dependency from scratch for each change that the test selection is run on. However, computing the

dependency graph is relatively fast. For example, computing it for MSL only takes 0.14% of the time it takes to run all tests. We would also like to detect changes on a more fine-grained level and identify which classes in a file that are actually changed.

Acknowledgements

We thank Jesper Öqvist for comments on an earlier draft of this paper. This research was partly supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA), within the strategic innovation program Process Industrial IT and Automation, under contract number (2017-02371).

References

- 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 2010a.
- Johan Åkesson, Torbjörn Ekman, and Görel Hedin. Implementation of a Modelica compiler using JastAdd attribute grammars. *Science of Computer Programming*, 75(1-2):21–38, January 2010b.
- Yih-Farn Chen, David S. Rosenblum, and Kiem-Phong Vo. Test-tube: A system for selective regression testing. In *Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, May 16-21, 1994.*, pages 211–220, 1994.
- Richard A. DeMillo, Richard J. Lipton, and Frederick G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11(4):34–41, 1978.
- Milos Gligoric, Lamyaa Eloussi, and Darko Marinov. Practical regression test selection with dynamic file dependencies. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, Baltimore, MD, USA, July 12-17, 2015*, pages 211–222, 2015.
- Erik Hedblom and Kasper Rundquist. Safe test selection for modelica using static analysis. Master’s thesis, Lund University, 2017. LU-CS-EX 2017-26.
- Görel Hedin and Eva Magnusson. JastAdd: an aspect-oriented compiler construction system. *Science of Computer Programming*, 47(1):37–58, 2003. ISSN 0167-6423. doi:[http://dx.doi.org/10.1016/S0167-6423\(02\)00109-0](http://dx.doi.org/10.1016/S0167-6423(02)00109-0).
- Modelica. *The Modelica Association*, 2018. <http://www.modelica.org>.
- Markus Olsson and Filip Stenström. Improved precision and verification for test selection in Modelica. Master’s thesis, Lund University, 2018. LU-CS-EX 2018-08.
- Jesper Öqvist, Görel Hedin, and Boris Magnusson. Extraction-based regression test selection. In *Proceedings of the 13th International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools, Lugano, Switzerland, August 29 - September 2, 2016*, pages 5:1–5:10, 2016.
- Gregg Rothermel and Mary Jean Harrold. Analyzing regression test selection techniques. *IEEE Trans. Software Eng.*, 22(8):529–551, 1996.
- Gregg Rothermel and Mary Jean Harrold. A safe, efficient regression test selection technique. *ACM Trans. Softw. Eng. Methodol.*, 6(2):173–210, 1997.
- Shin Yoo and Mark Harman. Regression testing minimization, selection and prioritization: a survey. *Softw. Test., Verif. Reliab.*, 22(2):67–120, 2012.

Functional Mock-up Interface: An empirical survey identifies research challenges and current barriers

Gerald Schweiger¹ Cláudio Gomes² Georg Engel¹ Irene Hafner³ Josef-Peter Schoegg⁴
Alfred Posch⁵ Thierry Noudui⁶

¹Technical University of Graz, Graz, Austria {gerald.schweiger, georg.engel}@tugraz.at

²University of Antwerp, Antwerp, Belgium claudio.gomes@uantwerp.be

³dwh GmbH - Simulation Services und Technical Solutions, Vienna, Austria irene.hafner@dwh.at

⁴KTH Royal Institute of Technology, Stockholm, Sweden schogg1@kth.se

⁵University of Graz, Graz, Austria alfred.posch@uni-graz.at

⁶Lawrence Berkeley National Laboratory, Berkeley, USA tsnoudui@lbl.gov

Abstract

Co-simulation is a promising approach for the analysis of complex, multi-domain systems, that leverages mature simulation tools of the respective domains. It has been applied in many different disciplines in academia and industry, with limited sharing of findings. With the increasing adoption of the FMI standard, researchers have set to work on surveying the scattered knowledge on co-simulation in academia. This paper complements the existing surveys by taking on the social and empirical aspect, corroborating, and prioritizing, previous findings. We focus on understanding the perceived research challenges, and the current barriers, based on expert assessment. One of the main barriers pointed out is the limited support for discrete event and hybrid co-simulation.

Keywords: Co-Simulation, Functional Mock-Up Interface, Modelling

1 Introduction

As engineered systems become more complex, whole system simulation techniques need to keep up with the increasing plethora of tools used in the development process. It is no longer reasonable to expect the existence of a one-size-fits-all modelling and simulation tool, capable of reproducing the behavior of a complex heterogeneous system, across the many development stages (Van der Auweraer et al., 2013; Vangheluwe et al., 2002). Instead, highly specialized modelling and simulation tools, each tailored to the needs of a specific engineering domain through years of research and development, should be integrated, to allow engineers to glimpse at the inter domain interactions of a coupled system.

For simulation, this integration can in theory be performed by describing how each of the models are translated to a uniform behavioral model, as suggested in (Vangheluwe, 2008). However, the existence of specialized suppliers with valuable Intellectual Property (IP), the subtleties of accurately simulating some formalisms, the sheer number of different modelling and simulation tools

and accompanying licensing fees, make this approach, denoted as co-modelling, infeasible in practice.

A pragmatic solution, called co-simulation (Gomes et al., 2018b; Hafner and Popper, 2017), is to perform the model integration at the dynamic behavioral level, where each model is used to produce a black box that consumes inputs and produces outputs over time. These black boxes, each representing the behavior of a subsystem/domain, can then be interconnected to mimic the interconnections of the corresponding subsystems. These interconnections frequently form feedback loops, which means that the behavior of one black-boxes, up to a simulated time point t , is only specified when the behavior of all the other interacting black-box has been computed up to t . The consequence is that the behavior of each black box must be computed in lock-step with the other black boxes, through the aid of a master algorithm. The master algorithm is responsible for: finding the appropriate initial values for every black-box; coordinating the progression of the simulated time; obtaining outputs and feeding inputs from/to the black-boxes; and instructing each black box to compute the next set of outputs. The algorithm is oftentimes summarized in time diagrams such as the one shown in Figure 1.

Co-simulation yields multiple advantages:

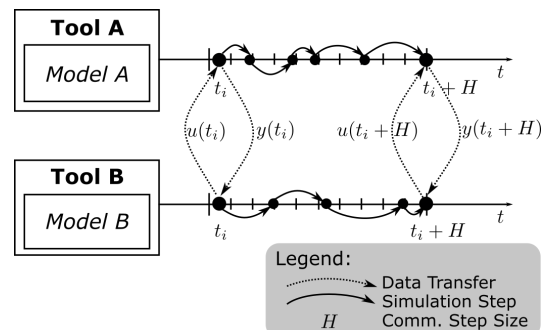


Figure 1. Example master algorithm.

- The behavioral level seems to be the simplest level any subsystem integration can be done, and is common across all behavioral formalisms;
- Each black box incorporates its own simulation algorithm, usually the most adequate for its domain;
- The exchange of the black box models can be made without requiring their content to be disclosed, thereby protecting IP, and avoiding licensing fees.

Unfortunately, naively connecting inputs to outputs on black boxes does not necessarily imply that the resulting behavior mimics the actual couplings of the subsystems, which brings us a main research problem in co-simulation: *are the co-simulation results trustworthy?*

This is not a new challenge, and the coupling of simulators can be traced back to multi-rate simulation techniques. However, the increasing number of applications in different domains (Schweiger et al., 2018a), have led researchers to survey the vast and scattered body of knowledge in co-simulation. For example, (Hafner and Popper, 2017) discusses the differences in terminology used regarding co-simulation. They provide a classification of existing co-simulation methods, which highlights the unexplored methods. With the intent of systematically surveying the academic state of the art, (Gomes et al., 2018b) introduces the fundamental concepts, and applies feature oriented domain analysis to construct a taxonomy of functional and non-functional requirements of co-simulation. This highlights the multiple ways in which information about the black-boxes can be exposed to attain more reliable results. The work in (Palensky et al., 2017) introduces the main concepts in co-simulation in a tutorial fashion. Despite its focus on power systems, it covers the main methods thoroughly, highlighting the pros and cons of each, and providing pointers to more detailed expositions.

To the best of our knowledge, even though co-simulation has been used in industry, there is no empirical assessment of its use, nor of the challenges described in the above surveys. Only (Bertsch et al., 2014) reports on the industrial use of co-simulation, and highlights some of the practical challenges in such a setting, but from the authors' experiences. There have been many other applications of co-simulation even since this report was published.

In this paper, we complement the existing survey work by taking on the social and empirical aspect. We collected interviews with international experts from various fields (both academic and industry) regarding applications, barriers and future challenges of Functional Mock-up Interface (FMI). The results presented here are part of a larger survey effort on co-simulation, whose results are still being collected. The FMI (Blockwitz et al., 2012; FMI, 2014) is a standard that enables co-simulation by providing a common interface to couple black box simulators. We focus on FMI based co-simulation, because of its adoption in various fields in industry and academia (Brem-

beck et al., 2011; Schweiger et al., 2018a; Bunte et al., 2014; Engel et al., 2018; Sanfilippo et al., 2018; Schweiger et al., 2018b) as well as increasing citations among scientific papers (see Figure 2).

In the next section, we describe our methodology, and in the section after, we summarize the main results and conclusions.

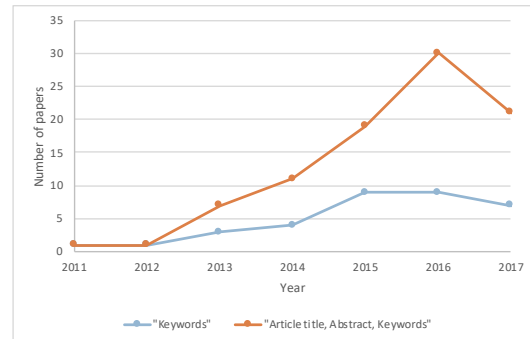


Figure 2. Example master algorithm.

2 Method

As a methodological foundation of this study, the Delphi method was adopted. The Delphi method is a forecasting technique with which the opinions from a defined group of experts are systematically collected and compiled (Hsu and Sandford, 2007). It enables the empirical investigation of research questions on topics that are characterized by an incomplete state of knowledge (Powell, 2003), a lack of historical data or a lack of agreement in the studied field (Okoli and Pawlowski, 2004). A Delphi study aims at achieving a reliable consensus of opinion, by conducting a repetitive assessment process that includes controlled opinion (Linstone and Turoff, 2002). As a formal consensus methodology, the Delphi method provides structured circumstances that “[...] can generate a closer approximation of the objective truth than would be achieved through conventional, less formal, and pooling of expert opinion” (Balasubramanian and Agarwal, 2012). We considered this method because it is especially useful for addressing interdisciplinary research problems, where the experts' opinions are heterogeneous

Regarding the number of experts, Clayton (1997) indicated that 15-30 experts with homogeneous expertise background or five to ten experts with heterogeneous background should be involved in a Delphi process, while Adler and Ziglio (1996) argued that 10-15 experts with homogeneous expertise can already be considered appropriate.

The quality of the Delphi process depends on the factors of creativity, credibility and objectivity (Nowack et al., 2011) and to address these quality criteria we followed acknowledged guidelines by authors such as (Landeta, 2006; Nowack et al., 2011; Okoli and Pawlowski, 2004).

For the selection of the sample of participants, we used a Knowledge Resource Nomination Worksheet (KRNW)

as a framework (Okoli and Pawlowski, 2004). The KRNW is a general criterion for sampling an expert panel to be included in a group technique study and consists in the following five steps (Delbecq et al., 1975): (1) Preparation of the KRNW; (2); Population of the KRNW; (3) Nomination of additional experts; (4) Ranking of experts; and (5) Invitation of experts (Okoli and Pawlowski, 2004).

In step 1, experts from academia and industry were selected, as we considered both perspectives essential. In step 2, the category academia was populated based on a keyword-based search in the relevant literature. The category of industry experts was compiled based on a keyword-based search in the relevant literature, the experience of the research group and consultation with practitioners. In step 3, both categories were expanded, based on the suggestions received after contacting the initial list of experts. The ranking of experts in step 4 was based on the number of publications (www.scopus.com). In step 5 the final list of 15 experts was invited to take part in the first phase of the Delphi study via an online-questionnaire.

The survey consist of two rounds. The choice of rounds is justified by, for instance, Sommerville, which argues that the changes in the participants' views in most cases occurred in the first two rounds of the study and not many new insights are gained on further rounds (Somerville, 2008). Table 1 summarizes the aim and approach of each round and provides the number of participants per category.

Table 1. Summary of the 2-stage Delphi process. Participants A=Academia, I=Industry, ND=not declared.

| Round | Aim | Approach | Participants | | | |
|-------|---|-------------------|--------------|----|----|-------|
| | | | A | I | ND | Total |
| 1 | Identification of research needs, SWOT factors, limitations and possible extensions of the FMI standard | Qualitative | 7 | 2 | 3 | 12 |
| 2 | Evaluation of the result from the first round and developing in-depth discussions on the key aspects. | Semi-quantitative | 17 | 11 | 0 | 28 |

Relevant questions regarding FMI in the first round were selected based on existing literature studies (e.g. (Gomes et al., 2018b; Palensky et al., 2017; Trcka et al., 2007) and the experience of the authors. Both rounds included both open-ended (qualitative) and quantitative questions.

In the first round, the majority of questions was qualitative, whereas in the second, quantitative. This ensures that the topic is introduced in a general way in the first round. If the first round consisted only of quantitative questions, there would be an increased risk of overlooking important factors or biasing the results.

The qualitative questions in the first round concerned only with findings that were common across the survey papers referred above. In these cases, expert opinions were used to evaluate findings in previous surveys and to enable quantitative statements and comparisons (e.g. how impor-

tant is the extension of the FMI standard in area “a” versus “b”).

The quantitative questions in the second round were mainly formulated based on the results of the first round and the findings in recent literature (e.g. when contradictions were identified).

A total of 28 experts answered the FMI relevant questions presented in this paper. Experts from academia who took part in the survey, work in the following fields: Software development, Energy Systems, Mobility and Maritime. Experts from industry, who took part in the survey, work in the following sectors: Energy Systems, Software development, Mobility. Some experts did not provide information about their field or sector.

A seven-point Likert scale was used to measure the quantitative responses (Entirely agree =7 to Entirely disagree = 1). In order to provide a transparent presentation of the results, (i) in the appendix, all results are displayed in detail in a bar chart and (ii) in Section 3 we present a summary table including Mean, Median and Interpolated Median values (Balasubramanian and Agarwal, 2012; Hallowell and Gambatese, 2010; Sachs, 1997)). There is an ongoing discussion about the best way to interpret Likert scales; Sachs argues that the interpolated median is more precise than the normal medians because of better consideration of frequencies of answers within one category in comparison to all answers (Sachs, 1997).

3 Results and Discussion

Table 2 summarizes the results from the second round of quantitative questions; more details can be found Figure 3.

The questions focus on the issues reported by the experts in the first round of the survey, and on the exiting literature. Based on the score provided by the experts to each question, we classified each issue according whether it constitutes a barrier for the adoption of the standard: issues with a median score less than 4 are considered as “Not a barrier”; issues with a median score between 4 and 5 are considered as “Somewhat of a barrier”; and issues with a median score of 5 or higher are considered as a “Barrier”.

For example, concerns with IP protection, with a median score of 3.0, do not constitute a barrier for the adoption of FMI. This corroborates the fact that one of the goals of FMI is to provide adequate IP protection (Blochwitz et al., 2011). This result does not necessarily contradict what is stated in (Durling et al., 2017), as that work concerns advanced use cases of co-simulation, such as design space exploration, or solving boundary conditions. As the authors suggest, it is likely that advanced co-simulation methods, or those providing formal guarantees (e.g., (Thule et al., 2018)), will require some information from the models.

We also tested the results on disagreement between experts from academia and industry using a Chi-square test. We found disagreement for the question: "There is a

lack of (scientific) community, forums, groups" ($p < 0.05$). Whereas the majority of industry experts did not consider it a barrier (median=3), experts from academia provided mixed answers (median=4).

In the following, we discuss the issues that experts consider to be barriers.

3.1 FMI has limited support for hybrid and discrete time co-simulation

Informally, a hybrid co-simulation is the co-simulation of a hybrid system (cf. (Gomes et al., 2017) for more details and examples). Hybrid systems exhibit a mix of continuous and discrete event dynamics; e.g., systems modelled with hybrid automata (Henzinger, 2000), switched systems (Sun, 2006), etc.

The ability to reproduce the dynamics of these systems in a co-simulation is important because, in full system evaluations, where co-simulation is frequently used (Van der Auweraer et al., 2013), hybrid dynamics are pervasive. For example, systems exhibiting Coulomb friction and/or hysteresis, or comprising non-trivial control software, all exhibit hybrid behavior.

In the FMI for co-simulation, version 2.0 (FMI, 2014), some support is provided to locate discontinuous events. However, according to the covered literature, providing support for hybrid co-simulation includes addressing the following challenges:

- Sound representation of different semantics (as done in (Ptolemaeus, 2014; Cremona et al., 2016) and semantic adaptations (Gomes et al., 2018a);
- Accurate event location (e.g., as done in (Zhang et al., 2008; Broman et al., 2013));
- Discontinuity identification and signal distinction (e.g., using the super-dense integer time formalization (Broman et al., 2015; Cremona et al., 2017a), or explicitly representing internal clocks (Franke et al., 2017); and
- Adequate discontinuity handling (e.g., set the internal continuous numerical solvers' state (Andersson et al., 2016)).

3.2 There is insufficient documentation

Detailed documentation, tutorials and examples are of central importance for the establishment of a technology such as co-simulation. Previous works have already addressed this barrier. (Palensky et al., 2017) presents a good introduction for researchers looking to understand the main co-simulation algorithms, and what their trade-offs are.

It is also important to mention that some tutorials have been published on individual standards or in the context of co-simulation projects. Within the the INTO-CPS (Larsen et al., 2016) project, for example, tutorials with industrial case studies were developed and training schools were organized. There are also tutorials for the FMI standard (FMI, 2018); some tool vendors also provide video tutorials on social media platforms such as Youtube.

The revision and/or introduction of online learning material based on insights into success factors in online education would be helpful (Volery and Lord, 2000; Sun et al., 2008). This should include real-world examples from different fields. Furthermore, the possibilities, problems and limitations of applications in the field of continuous, discrete event and hybrid co-simulation should be presented. In order to sustain a long term adoption of the standard and to lower the entry barrier for new user, it is important to manage expectations of what co-simulation can, and cannot, do. This includes e.g. licensing issues, computational performance in comparison to monolithic simulations. The integration of FMI into university courses would increase the visibility of the standard and accelerate the development of (online) learning materials and tutorials.

3.3 The standard does not support certain requirements that would be widely needed by industry and academia

The authors are aware that this statement is very general and answers based on Liker Scales do not allow general conclusions; several extensions to the standard have been proposed from tool vendors (e.g. (Sahlin and Lebedev, 2016)), industry (e.g. (Hirano et al., 2015) and academia (e.g. (Cremona et al., 2017b; Broman et al., 2013))). Some of these proposed extensions are addressed in the current development process (FMI, 2018). In addition to the ongoing FMI development process, we propose a comprehensive empirical study to clarify which extensions are needed by which actors in industry and academia. In this context, one expert pointed out that if all extensions and peculiarities of individual tools are considered, there is a risk that the robustness of applications will be reduced. Therefore, the proposed empirical study should also include theoretical experts, tool and members of the FMI development committee.

3.4 Lack of transparency in in features supported by FMI tools

Potential users usually have a clear idea of the modeling requirements when addressing a problem with co-simulation. Based on these requirements, a screening of possible alternatives often follows. A transparent and easy-to-understand presentation of supported features is of central importance in this context. We propose two actions: (i) which features are supported, and which are not, should also be addressed in online learning materials and tutorials (see section 3.2); and (ii) a transparent and frequently updated online presentation of supported features and planned extensions.

3.5 Limitations of the study

The aim of this study is to identify barriers to FMI by means of empirical surveys and to link and critically reflect on findings from recent literature. How these barriers could be overcome was also discussed in relation to re-

Table 2. Expert assessment of current barriers for FMI based on a Seven-point Likert scale.

| | Mean | Median | Interpolated Median |
|--|------|--------|---------------------|
| Not a Barrier | | | |
| It is difficult to post-process simulation results | 3.57 | 2.50 | 2.50 |
| Concerns of industry/academia regarding FMI and IP protection | 3.52 | 3.00 | 2.83 |
| No pre-implemented Master Algorithms | 4.08 | 3.00 | 3.25 |
| Somewhat of a Barrier | | | |
| The FMI-standard still requires a number of updates in order to serve as a useful general standard for co-simulation | 4.52 | 4.00 | 3.75 |
| There is not enough cooperation and exchange (theoretical/numerical, implementation, application/industry) in defining and developing the FMI standard | 4.12 | 4.00 | 3.81 |
| There is a lack of tools that sufficiently support FMI | 4.04 | 4.00 | 3.83 |
| There is a lack of (scientific) community, forums, groups | 4.27 | 4.00 | 3.83 |
| Simulations are slow compared to monolithic simulations | 3.82 | 4.00 | 3.92 |
| It is difficult to implement FMU's (API, connecting/linking different subsystems) | 4.07 | 4.00 | 4.00 |
| Barrier | | | |
| FMI has limited support for hybrid co-simulation and it is not easily applicable | 5.82 | 5.00 | 5.00 |
| Lack of transparency in features supported by FMI tools | 5.12 | 5.00 | 5.05 |
| There is insufficient documentation and a lack of examples, tutorials, etc. | 5.14 | 5.00 | 5.17 |
| The standard does not support certain requirements that would be widely needed by industry and academia | 5.42 | 5.00 | 5.25 |
| FMI has limited support for discrete co-simulation and it is not easily applicable | 5.67 | 5.00 | 5.25 |

cent literature. The identification of new approaches and the quantitative and qualitative evaluation and comparison of existing approaches for the respective barriers is beyond the scope of this paper.

The barrier *"The standard does not support certain requirements that are urgently needed by industry and academia"* is very general and a detailed discussion goes beyond the scope of this paper. The authors admit that ideally, experts should have been asked in detail about these requirements. Nevertheless, we did not want to withhold these results, as they could stimulate a broader discussion on that topic.

A further limitation of the present study concerns the size of the sample. However, the aim of Delphi studies is not to obtain a representative sample in a purely statistical sense. The number of experts participating in this study is in line with recommendations from relevant literature on Delphi studies (Adler and Ziglio, 1996; Clayton, 1997; Ludwig, 1997). A general critical discussion about the Delphi method and its weaknesses can be found here (Goodman, 1987; Hill and Fowles, 1975).

4 Conclusion

The present paper reports an expert assessment on FMI, taking on the social and empirical aspect, with a focus on understanding the perceived research challenges and the current barriers. After a two-round Delphi-method, we

concluded that experts consider the following as barriers to the adoption of the standard:

1. limited support for hybrid- and discrete event co-simulation;
2. insufficient documentation and a lack of examples and tutorials;
3. lack of certain requirements that would be widely needed by industry and research; and
4. transparent presentation of supported features;

It is our hope that the results of this study increase transparency and facilitate a structured development of the standard, and related research.

5 Acknowledgments

We want to thank all experts who participated in our study. The research was supported by ECSEL JU under the project H2020 737469 AutoDrive - Advancing fail-aware, fail-safe, and fail-operational electronic components, systems, and architectures for fully automated driving to make future mobility safer, affordable, and end-user acceptable. AutoDrive is funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2017 and April 2020. More information <https://iktderzukunft.at/en/>

References

- Michael Adler and Erio Ziglio. *Gazing Into the Oracle: The Delphi Method and Its Application to Social Policy and Public Health*. Jessica Kingsley Publishers, London and Philadelphia, 1996.
- Christian Andersson, Claus Führer, and Johan Åkesson. Efficient Predictor for Co-Simulation with Multistep Sub-System Solvers. Technical Report 1, 2016. URL <http://lup.lub.lu.se/record/dbaf9c49-b118-4ff9-af2e-e1e3102e5c22>.
- Ramya Balasubramanian and Deepti Agarwal. Delphi Technique- A Review. *International Journal of Public Health Dentistry*, 3(2):16–25, 2012. ISSN 17411645. URL <http://journalgateway.com/ijphd/article/view/444>.
- Christian Bertsch, Elmar Ahle, and Ulrich Schulmeister. The Functional Mockup Interface-seen from an industrial perspective. In *10th International Modelica Conference*, 2014.
- Torsten Blochwitz, Martin Otter, Martin Arnold, C Bausch, Christoph Clauss, Hilding Elmqvist, Andreas Junghanns, Jakob Mauss, M Monteiro, T Neidhold, Dietmar Neumerkel, Hans Olsson, J.-V. Peetz, and S Wolf. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In *8th International Modelica Conference*, pages 105–114, Dresden, Germany, 6 2011. Linköping University Electronic Press; Linköpings universitet. doi:10.3384/ecp11063105.
- Torsten Blockwitz, Martin Otter, Johan Akesson, Martin Arnold, Christoph Clauss, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauss, Dietmar Neumerkel, Hans Olsson, and Antoine Viel. Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In *9th International Modelica Conference*, pages 173–184, Munich, Germany, 11 2012. Linköping University Electronic Press. doi:10.3384/ecp12076173.
- Jonathan Brembeck, Martin Otter, and Dirk Zimmer. Non-linear Observers based on the Functional Mockup Interface with Applications to Electric Vehicles. *Proceedings of the 8th International Modelica Conference*, pages 474–483, 2011. doi:10.3384/ecp11063474. URL <http://www.ep.liu.se/ecp/article.asp?issue=063%26article=53>.
- David Broman, Christopher Brooks, Lev Greenberg, Edward A Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. Determinate composition of FMUs for co-simulation. In *Eleventh ACM International Conference on Embedded Software*, page Article No. 2, Montreal, Quebec, Canada, 2013. IEEE Press Piscataway, NJ, USA. ISBN 978-1-4799-1443-2.
- David Broman, Lev Greenberg, Edward A Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. Requirements for Hybrid Cosimulation Standards. In *18th International Conference on Hybrid Systems: Computation and Control*, HSCC '15, pages 179–188, Seattle, Washington, 2015. ACM New York, NY, USA. ISBN 978-1-4503-3433-4. doi:10.1145/2728606.2728629.
- Tilman Bünte, Lok Man Ho, Clemens Satzger, and Jonathan Brembeck. Central Vehicle Dynamics Control of the Robotic Research Platform RoboMobil. *ATZelektronik worldwide*, 9(3):58–64, 6 2014. ISSN 2192-9092. doi:10.1365/s38314-014-0254-6. URL <https://doi.org/10.1365/s38314-014-0254-6>.
- Mark J Clayton. Delphi: a technique to harness expert opinion for critical decision-making tasks in education. *Educational Psychology*, 17(4):373–386, 1997. doi:10.1080/0144341970170401. URL <https://doi.org/10.1080/0144341970170401>.
- Fabio Cremona, Marten Lohstroh, Stavros Tripakis, Christopher Brooks, and Edward A Lee. FIDE: an FMI integrated development environment. In *31st Annual ACM Symposium on Applied Computing*, SAC '16, pages 1759–1766, Pisa, Italy, 2016. ACM New York, NY, USA. ISBN 9781450337397. doi:10.1145/2851613.2851677.
- Fabio Cremona, Marten Lohstroh, David Broman, Edward A. Lee, Michael Masin, and Stavros Tripakis. Hybrid co-simulation: It's about time. *Software & Systems Modeling*, November 2017a. ISSN 1619-1366, 1619-1374. doi:10.1007/s10270-017-0633-6.
- Fabio Cremona, Marten Lohstroh, David Broman, Stavros Tripakis, Edward A Lee, and Michael Masin. Hybrid Co-simulation: It's About Time. Technical report, Report No. UCB/EECS-2017-6, EECS Department, University of California, Berkeley, 2017b. URL <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-6.html>.
- André L Delbecq, Andrew H Van de Ven, and David H Gustafson. *Group techniques for program planning: A guide to nominal group and delphi processes*. Scott-Foresman and Company, Glenview, Illinois, 1975.
- Erik Durling, Elias Palmkvist, and Maria Henningsson. FMI and IP protection of models: A survey of use cases and support in the standard. In *12th International Modelica Conference*, number 132, pages 329–335. Linköping University Electronic Press, 2017. ISBN 1650-3740.
- Georg Engel, Ajay S. Chakkaravarthy, and Gerald Schweiger. A General Method to Compare Different Co-Simulation Interfaces: Demonstration on a Case Study. In Janusz Kacprzyk, editor, *Simulation and Modeling Methodologies, Technologies and Applications*, chapter 19. Springer, 2018. doi:10.1007/978-3-030-01470-4_19.
- FMI. Functional Mock-up Interface for Model Exchange and Co-Simulation. Technical report, 2014.
- FMI. Functional Mock-up Interface, 2018. URL <https://fmi-standard.org>.
- Rüdiger Franke, Sven Erik Mattsson, Martin Otter, Karl Wernersson, Hans Olsson, Lennart Ochel, and Torsten Blochwitz. Discrete-time models for control applications with FMI. pages 507–515, July 2017. doi:10.3384/ecp17132507.

- Cláudio Gomes, Yentl Van Tendeloo, Joachim Denil, Paul De Meulenaere, and Hans Vangheluwe. Hybrid System Modelling and Simulation with Dirac Deltas. Technical report, University of Antwerp, Antwerp, 2 2017. URL <http://arxiv.org/abs/1702.04274>.
- Cláudio Gomes, Bart Meyers, Joachim Denil, Casper Thule, Kenneth Lausdahl, Hans Vangheluwe, and Paul De Meulenaere. Semantic Adaptation for FMI Co-simulation with Hierarchical Simulators. *SIMULATION*, pages 1–29, 2018a. doi:10.1177/0037549718759775.
- Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: a Survey. *ACM Computing Surveys*, 51(3):Article 49, 4 2018b. doi:10.1145/3179993.
- Claire M. Goodman. The Delphi technique: a critique. *Journal of Advanced Nursing*, 12(6):729–734, 1987. ISSN 13652648. doi:10.1111/j.1365-2648.1987.tb01376.x.
- Irene Hafner and Niki Popper. On the terminology and structuring of co-simulation methods. In *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 67–76, New York, New York, USA, 2017. ACM Press. ISBN 9781450363730. doi:10.1145/3158191.3158203. URL <http://dl.acm.org/citation.cfm?doid=3158191.3158203>.
- Matthew R. Hallowell and John A. Gambatese. Qualitative Research: Application of the Delphi Method to CEM Research. *Journal of Construction Engineering and Management*, 136(1):99–107, 1 2010. ISSN 0733-9364. doi:10.1061/(ASCE)CO.1943-7862.0000137. URL <http://ascelibrary.org/doi/10.1061/%28ASCE%29CO.1943-7862.0000137>.
- Thomas A Henzinger. *The theory of hybrid automata*. Springer, 2000. ISBN 3642640524.
- Kim Quaille Hill and Jib Fowles. The methodological worth of the Delphi forecasting technique. *Technological Forecasting and Social Change*, 7(2):179–192, 1975. ISSN 0040-1625. doi:https://doi.org/10.1016/0040-1625(75)90057-8. URL <http://www.sciencedirect.com/science/article/pii/0040162575900578>.
- Yutaka Hirano, Satoshi Shimada, Yoichi Teraoka, Osamu Seya, Yuji Ohsumi, Shintaroh Murakami, Tomohide Hirono, and Takayuki Sekisue. Initiatives for acausal model connection using FMI in JSAE (Society of Automotive Engineers of Japan). In *Proceedings of the 11th International Modelica Conference*, pages 795–801, 2015. doi:10.3384/ecp15118795. URL http://www.ep.liu.se/ecp_article/index.en.aspx?issue=118;article=85.
- Chia-chien Hsu and Brian Sandford. The delphi technique: making sense of consensus. *Practical Assessment, Research & Evaluation*, 12(10):1–8, 2007. ISSN 1531-7714. doi:10.1016/S0169-2070(99)00018-7.
- Jon Landeta. Current validity of the Delphi method in social sciences. *Technological Forecasting and Social Change*, 73(5):467–482, 2006. ISSN 00401625. doi:10.1016/j.techfore.2005.09.002.
- Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, Peter Fritzon, Jorg Brauer, Christian Kleijn, Thierry Lecomte, Markus Pfeil, Ole Green, Stylianos Basagiannis, and Andrey Sadovykh. Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project. In *2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*, pages 1–6, Vienna, Austria, 4 2016. IEEE. ISBN 978-1-5090-1154-4. doi:10.1109/CPSData.2016.7496424.
- Harold A Linstone and Murray Turoff. The Delphi Method: Techniques and Applications. *Technometrics*, 18:363, 2002. ISSN 00401706. doi:10.2307/1268751.
- Barbara Ludwig. Predicting the Future: Have you considered using the Delphi Methodology? *Journal of Extension*, 35(5):5TOT2, 1997. ISSN 10775315. doi:10.1161/CIRCULATIONAHA.111.023879. URL <http://www.joe.org/joe/1997october/tt2.php>.
- Martin Nowack, Jan Endrikat, and Edeltraud Guenther. Review of Delphi-based scenario studies: Quality and design considerations. *Technological Forecasting and Social Change*, 78(9):1603–1615, 2011. ISSN 00401625. doi:10.1016/j.techfore.2011.03.006. URL <http://dx.doi.org/10.1016/j.techfore.2011.03.006>.
- Chitu Okoli and Suzanne D Pawlowski. The Delphi method as a research tool : an example , design considerations and applications. *Information & Management*, 42(1):15–29, 2004. ISSN 03787206. doi:10.1016/j.im.2003.11.002. URL <http://dx.doi.org/10.1016/j.im.2003.11.002>.
- Peter Palensky, Arjen A Van Der Meer, Claudio David Lopez, Arun Joseph, and Kaikai Pan. Cosimulation of Intelligent Power Systems: Fundamentals, Software Architecture, Numerics, and Coupling. *IEEE Industrial Electronics Magazine*, 11(1):34–50, 2017. ISSN 1932-4529. doi:10.1109/MIE.2016.2639825. URL <http://ieeexplore.ieee.org/document/7883974/>.
- Catherine Powell. The Delphi Technique: myths and realities. *Methodological Issues in Nursing Research*, 41(4):376–382, 2003. ISSN 0309-2402. doi:10.1046/j.1365-2648.2003.02537.x. URL <http://www.ncbi.nlm.nih.gov/pubmed/12581103>.
- Claudius Ptolemaeus. *System Design, Modeling, and Simulation: Using Ptolemy II*. Berkeley: Ptolemy.org, 2014. ISBN 1304421066.
- Lothar Sachs. *Angewandte Statistik*. Springer-Verlag, Berlin Heidelberg, 1997.
- Per Sahlin and Alexey Lebedev. OPENCPS: Benchmark building and energy system models. Technical report, 2016.
- Filippo Sanfilippo, Lars Ivar Hatledal, Kristin Ytterstad Pettersen, and Houxiang Zhang. A Benchmarking Framework for Control Methods of Maritime Cranes Based on the Functional Mockup Interface. *IEEE Journal of Oceanic Engineering*, 2018. ISSN 03649059. doi:10.1109/JOE.2017.2691920.

- Gerald Schweiger, Cláudio Gomes, Georg Engel, Irene Hafner, Josef-Peter Schoeggel, Alfred Posch, and Thierry Stephane Nouidui. An Empirical Survey on Co-Simulation: Promising Standards, Challenges and Research Needs. *Manuscript submitted for publication*, 2018a.
- Gerald Schweiger, Richard Heimrath, Basak Falay, Keith ODonovan, Peter Nageler, Reinhard Pertschy, Georg Engel, Wolfgang Streicher, and Ingo Leusbrock. District Energy Systems: Modelling paradigms and general-purpose tools. *Energy*, 2018b.
- Jerry Somerville. Critical Factors Affecting the Assessment of Student Learning Outcomes: A Delphi Study of the Opinions of Community College Personnel. *Journal of Applied Research in the Community College*, 15(2):109–119, 2008. ISSN 1068-610X.
- Pei-Chen Sun, Ray J Tsai, Glenn Finger, Yueh-Yang Chen, and Dowming Yeh. What drives a successful e-Learning? An empirical investigation of the critical factors influencing learner satisfaction. *Computers & Education*, 50(4):1183–1202, 2008. ISSN 0360-1315. doi:<https://doi.org/10.1016/j.compedu.2006.11.007>. URL <http://www.sciencedirect.com/science/article/pii/S0360131506001874>.
- Zhendong Sun. *Switched linear systems: control and design*. Springer Science & Business Media, 2006. ISBN 1846281318.
- Casper Thule, Cláudio Gomes, Julien Deantoni, Peter Gorm Larsen, Jörg Brauer, and Hans Vangheluwe. Towards Verification of Hybrid Co-simulation Algorithms. In *2nd Workshop on Formal Co-Simulation of Cyber-Physical Systems*, page to be published, Toulouse, France, 2018. Springer, Cham.
- Marija Trcka, Michael Wetter, and Jan Hensen. Comparison of co-simulation approaches for building and HVAC/R system simulation. In *International IBPSA Conference*, Beijing, China, 2007.
- Herman Van der Auweraer, Jan Anthonis, Stijn De Bruyne, and Jan Leuridan. Virtual engineering at work: the challenges for designing mechatronic products. *Engineering with Computers*, 29(3):389–408, 2013. ISSN 0177-0667. doi:10.1007/s00366-012-0286-6.
- Hans Vangheluwe. Foundations of Modelling and Simulation of Complex Systems. *Electronic Communications of the EASST*, 10, 2008. doi:10.14279/tuj.eceasst.10.162.148.
- Hans Vangheluwe, Juan De Lara, and Pieter J Mosterman. An introduction to multi-paradigm modelling and simulation. In *AI, Simulation and Planning in High Autonomy Systems*, pages 9–20. SCS, 2002.
- Thierry Volery and Deborah Lord. Critical success factors in online education. *International Journal of Educational Management*, 14(5):216–223, 9 2000. ISSN 0951-354X. doi:10.1108/09513540010344731. URL <http://www.emeraldinsight.com/doi/10.1108/09513540010344731>.
- Fu Zhang, Murali Yeddanapudi, and Pieter J Mosterman. Zero-Crossing Location and Detection Algorithms For Hybrid System Simulation. In *IFAC Proceedings Volumes*, volume 41, pages 7967–7972, Seoul, Korea, 7 2008. Elsevier Ltd. doi:10.3182/20080706-5-KR-1001.01346. URL <http://linkinghub.elsevier.com/retrieve/pii/S1474667016402296>.

6 Appendix

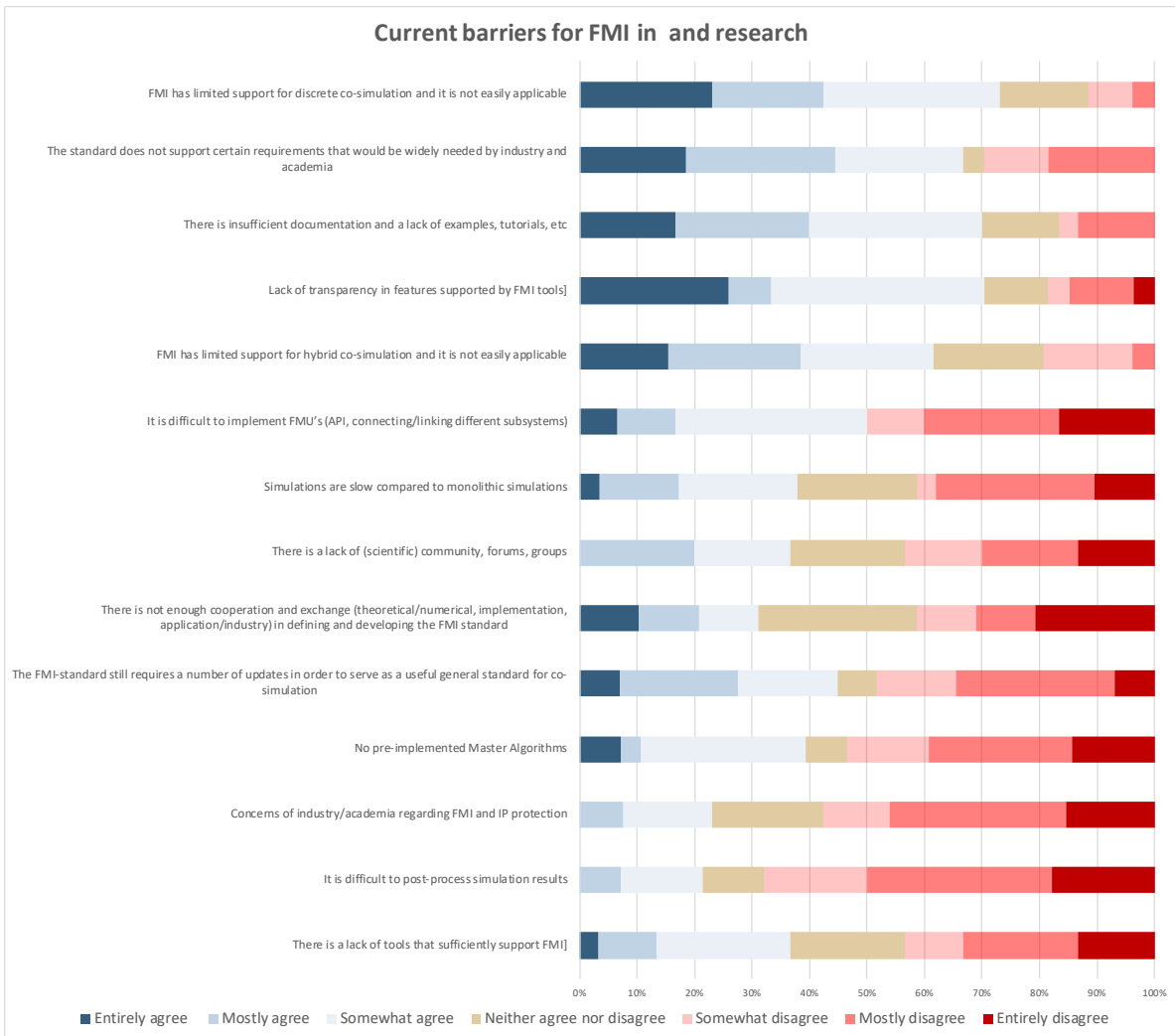


Figure 3. Expert assessment of current barriers for FMI based on a Seven-point Likert scale.

A Method to Import an FMU to a Hardware Description Language

Min Zhang

Synopsys Inc., USA, minz@synopsys.com

Abstract

In this paper, a new method of importing FMUs (Functional Mock-up Unit) [1] to a multi-domain, mixed-mode simulator is presented. Supporting FMI (Functional Mock-up Interface) 2.0 for Model Exchange by converting an FMU to an HDL (Hardware Description Language) wrapper model not only takes advantage of the existing simulator capabilities, but also avoids a significant amount of work in the core of the simulator. The selected HDL in this paper is MAST which is used in both Saber and SaberHDL simulators [2][7]. To make the FMU import process easier, a general conversion utility, FMU2MAST, was developed which converts an FMU to a MAST model automatically. Two examples, bouncing ball and motor drive system are presented. With these two examples, three techniques used in this method are discussed: Accurate event detection in a variable time-step integration algorithm; Re-initialization of a state variable in MAST; and solving DAE (Differential Algebraic Equation) of a coupling FMUs system. This new FMU import method has been proved a success with 44 examples exported from five different tools.

Keywords: FMI, FMU, HDL, MAST, Modeling, Simulation, Saber, SaberHDL, DAE

1 Introduction

In order to improve the exchange of simulation models between suppliers and OEMs, FMI (Functional Mock-up Interface) is initiated by Daimler AG in 2010. It defines an interface to be implemented by an executable called FMU (Functional Mock-up Unit). It has been used in automotive and non-automotive industries, and supported by many simulation tools [1].

Saber is a multi-domain, mixed-mode simulator used in automotive and aerospace industries for more than thirty years. It supports models written by an HDL such as MAST and VHDL-AMS [4][7]. In recent years, there are more and more requests to import FMUs into Saber simulator from the industry.

To support the FMI 2.0 for Model Exchange in an HDL simulator, it requires a significant work in the core of the simulator. In this paper, a new method to import FMUs

for Model Exchange is explored. Instead of supporting the FMU in a simulator core, the method proposed in this paper is to convert an FMU into a MAST wrapper model. This method has the following advantages:

1. Reduce significant work in the core of the simulator, which is time consuming as well as risky.
2. Avoid the duplicated work in another MAST simulator to support FMU import. Once an FMU is successfully converted to a MAST model, the generated MAST model works in Saber simulator, it also can work in another MAST simulator, SaberHDL, without any extra work.
3. The generated MAST wrapper model inherits all the features of the MAST language, and is applicable for all the existing simulator analyses, such as operating analysis, transient analysis, and advanced Monte Carlo analysis [7].
4. The generated MAST wrapper models can be used along with other models written in MAST or VHDL-AMS [4][6]. It increases the model availability and helps the study of a more complex and interesting system.

In chapter 2, a detail analysis of the data types and variables in FMU and MAST language is presented. Based on that, the equivalent objects in MAST language is derived. A new interface, saberFMI is introduced in chapter 3. It creates the communication channel between MAST models and FMUs through FMI interfaces. In chapter 4, A general conversion utility, FMU2MAST is introduced. It parses the model description file in the FMU, generates an equivalent MAST wrapper model automatically, which makes the conversion from FMUs to MAST models easier. Two FMU examples are presented in the chapter 5 and 6 to discuss the detail techniques used in the method. Although the bouncing ball example is simple, it is useful to discuss two issues: accurate event detection in variable-time step integration algorithm and re-initialization of a state variable in MAST model. The motor drive system is used to illustrate how a coupling system with multiple FMUs are solved in a DAE solver. The method has been verified in two simulators with 44 FMU examples exported by five different tools.

2 Model Types and Variables

Both FMUs and MAST models are designed to describe a mixed-mode, multi-physics system, consequently they share many interchangeable objects, which are discussed below [1][2].

2.1 Data Types

MAST has six data types. The four basic scalar data types are integer, number, string and enumeration. These types have the same definition as the data types defined in the FMUs. The difference is that the identifier name in the FMUs is case sensitive, and it also can have special characters, such as space, parenthesis, braces and brackets in it, e.g. “der(v)”. In such cases, an underscore “_” is used to replace the illegal MAST characters, therefore, the variable name “der(v)” in the FMU will be translated into “der_v” in MAST. Due to lack of char and byte types in MAST, any FMUs with these variable types will not be supported for MAST conversion.

2.2 Model Connection Points

In MAST, the model connection points communicate the characteristics of the model with the rest of the system. There are three different types of connection points: continuous analog pin, event-driven port and data flow type.

The continuous analog data type presents the physical connection which has across and through units. For example, the electrical port has voltage as a cross unit and current as a through unit. The across and through units satisfy the KCL (Kirchhoff's Voltage Law) and KVL (Kirchhoff's Current Law) laws, thus the analog pin is energy conservative. There is no direction for this type. Currently FMI 2.0 doesn't support physical connection port. A method has been proposed to create an adaptor model for the physical port to solve the problem [8]. The proposal assumes the voltage as input, and current as the output, which may not applicable for all the cases. The better way is to solve the equations associated with the physical ports in a DAE solver, which may be supported in FMI/FMU standard in the future.

The event-driven port is used to communicate a model's discrete behavior in the system. It has three direction modes: input, output and inout (also called bi-direction). It is the input mode if it is driven by other discrete events and output mode if it drives other models. It is the inout mode if it adopts both behaviors. It is equivalent to the FMU scalarVariable with the variability of discrete and causality of input or output.

The data flow connection describes a model in a control flow fashion. It has input and output modes. It is input mode if it reads values from the connection point and output mode if it writes values to the connection. It is

equivalent to the FMU scalarVariable with the variability of continuous and causality of input or output.

2.3 Model Parameter

Model parameters are coefficients that reside within physical characteristic equations which describe the model behavior. During simulation, the model parameters remain constant, however it can be varied in different simulation runs. The model parameter is equivalent to the FMU scalarVariable with the causality of parameter and variability of constant.

2.4 Constant Variable

The constant variable is similar to the model parameter, but used locally and only visible inside the model. Its value is constant, or may be calculated based on other model parameters but remains constant during the simulation. The constant variable in MAST model is equivalent to the FMU scalarVariable with the following types: 1. Causality is parameter and variability is fixed; 2. Causality is calculatedParameter and variability is fixed or constant; 3. Causality is local and variability is fixed or tunable.

2.5 State Variable

In MAST, the state variable (**state**) is used to describe the discrete behavior whose value remains constant between two consecutive time steps but may change from time point to time point. It is equivalent to the FMU scalarVariable with the following types: 1. Causality is independent; 2. Causality is local and variability is either discrete or tunable; 3. Causality is parameter and variability is tunable.

2.6 Local Analog Variable

A local analog variable in MAST (**val**) is a continuous variable and used to simplify the complicated system equations. It is equivalent to the FMU variable with the causality of local and variability of continuous.

2.7 System Analog Variable

In MAST, system variables (**var**) are the unknown variables that are needed to be simultaneously solved via the DAE (Differential Algebraic Equations) solver. Usually they are the analog connection points, data flow connections, through variable of independent source and **d_by_dt** operators for differential equations. FMU only solves the ODE (Ordinary Differential Equations), thus MAST system variable (**var**) is equivalent to an FMU continuous state variable with the causality of local and variability of continuous. The exchangeable objects between MAST and FMUs are shown in figure 1.

| | | | FMI | | | |
|------|------------|------------|-----------|---------------------|----------|--|
| | | | causality | variability | state | |
| MAST | connection | input | input | continuous | | |
| | | output | output | continuous | | |
| | | state | | | | |
| | variable | (in) | input | input | discrete | |
| | | | output | output | discrete | |
| | | parameter | parameter | parameter | fixed | |
| | | | constant | calculatedParameter | fixed | |
| | | | | local | fixed | |
| | | state | | local | constant | |
| | | | | parameter | tunable | |
| | | | | independent | | |
| | | | | local | discrete | |
| | local | tunable | | | | |
| val | local | continuous | | | | |
| var | local | continuous | yes | | | |

Figure 1 Exchangeable objects between the MAST and FMU

3 saberFMI Interface

In order to communicate with an FMU inside a MAST model, a new MAST foreign routine interface, `saberFMI` is developed. It exchanges the information between the MAST and FMU through the FMI. With this interface, the simulator can talk to FMUs through the interfaces: MAST, `saberFMI` and FMI. The communication interfaces between the FMUs and Saber simulator are shown in figure 2.

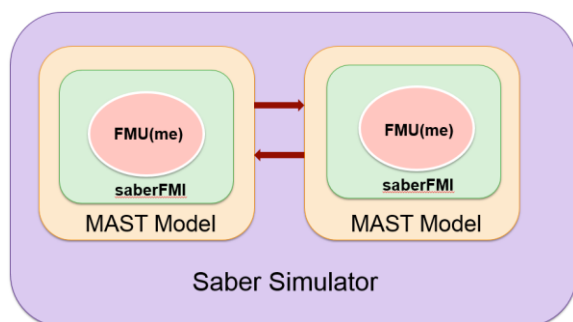


Figure 2 Interfaces between Simulator and FMUs

3.1 Parameter Section

The **Parameters** section is a group of sequential statements that initializes the system. It validates all the model parameters, calculates the internal constant variables which depend on the model parameters, and initializes state variables if needed. It is equivalent to **instantiated** and **initializationMode** states defined in FMI state machine [1]. Four `saberFMI` interfaces are introduced in this section: `initialization`, `setValues`, `updateValues` and `getValues`. The first `initialization` `saberFMI` interface calls `fmi2Instantiate` to construct the `fmi2Component` and return `fmiHandle` to the MAST model. The second `setValues` interface passes all the model parameters to the FMU by calling `fmi2SetX` internally (where **X** is one of the FMU data types, e.g. Integer, Real and Boolean). The third `updateValues`

interface will update all the internal variables by calling the FMI interface `fmi2EnterInitializationMode` and `fmi2ExitInitializationMode`. The fourth `saberFMI` interface, `getValues`, will get all the internal local variables by calling `fmi2GetX` and pass them back to the MAST model. The `saberFMI` calling sequence in MAST model during the initialization phase is shown in figure 3.

```
Parameters {
  fmiHandle = saberfmi(initialization...) {
    fmi2Instantiate();
  }
  saberfmi(setValues,fmiHandle,...) {
    fmi2setReal/Integer/Boolean();
  }
  saberfmi(updateValues,fmiHandle...) {
    fmi2EnterInitializationMode();
    fmi2ExitInitializationMode();
  }
  h_ic = saberfmi(getValues,fmiHandle...) {
    fmi2GetReal/Integer/Boolean();
  }
}
```

Figure 3 `saberFMI` in MAST Parameter section

3.2 When Sections

The **when** section in MAST is used to construct the discrete state machine. Once the input events get changed, Saber simulator will call the statements in the body of the **when** section, and propagate the events until no new events are generated. This section is equivalent to the **EventMode** state in the FMI state machine. Two new `saberFMI` interfaces are introduced: `setEvents` and `checkEvents`. The first `setEvents` interface will call `fmi2EnterEventMode`, then `fmi2SetX` to update input

```
When(event_on(in1)) {
  saberfmi(setEvent,fmiHandle,...) {
    fmi2EnterEventMode();
    fmi2setReal/Integer();
  }
  nextEvent=saberfmi(checkEvents,fmiHandle,...) {
    while(newDiscreteStatesNeeded) {
      fmi2newDiscreteStates();
    }
    fmi2EnterContinuousTimeMode();
  }
  schedule_next_time(nextEvent)
}
```

Figure 4 `saberFMI` in MAST when section

discrete variables. Next, **fmi2NewDiscreteStates** will be repeatedly called to propagate all the events until no new event is generated. The FMI interface **fmi2EnterContinuousTimeMode** will be called later to appropriately switch back to **continuousTimeMode**. At the end of **when** section, the MAST built-in scheduling function, **schedule_next_time()**, will be called to force the next simulation time to be the value returned by *checkEvents*. The FMI calling sequence in MAST **when** section is shown in figure 4.

3.3 Values Section

In MAST, the **values** section of a model is used to transform variables into a form needed in the equations section. In this section, three new saberFMI interfaces are introduced: *timeValue*, *nonlinear* and *checkCross*. The first one, *timeValue*, passes continuous input values to the FMU by calling **fmi2SetTime** and **fmi2SetReal**. The second saberFMI interface, *nonlinear*, receives the nonlinear function values of the scalarVariables defined in the FMU by calling **fmi2GetReal**. Saber simulator will automatically construct the nonlinear functions, extract the numerical partial derivatives for each nonlinear dimension, and fill in system Jacobin matrix. The third saberFMI interface, *checkCross*, will get eventIndicator value, return zero if no cross event is detected, and one a cross event is detected. This evenIndicator will be used in a **when (threshold)** section to find the exact time when any event occurs and force the simulator to find a solution at that time point. All these steps are accomplished in the **continuousTimeMode** state. The FMI calling sequence in MAST **values** section are shown in figure 5.

```

Values {
  saberfmi(timeValue,fmiHandle,input...) {
    fmi2setReal ();
  }
  xIndicator=saberfmi(checkCrosss,fmiHandle,time...) {
    fmi2getEventIndicators();
  }
  der_h=saberfmi(nonlinear,fmiHandle,time,h,v) {
    fmi2setReal();
    fmi2completedIntegratorStep();
    fmi2getReal();
  }
}

```

Figure 5 saberFMI in MAST values section

4 FMU2MAST conversion

With the new saberFMI interface introduced in section 3, the FMU models can be represented by a MAST wrapper model and simulated in Saber simulator. It is nevertheless very challenging to translate an FMU to a

MAST model manually because it requires advanced knowledge of both FMU and MAST. To assist the conversion process, a new utility FMU2MAST was created.

The conversion process can be divided into three steps. The first step is to read in the model description file, parse the XML file, and build up the XML tree in the memory. The second step is to preprocess the variables. First, all the illegal MAST variables will be renamed to valid MAST names; Next, the alias variables, which share the same **ValueReference** attribute, will be identified. The alias variables can be represented in MAST model with a simple assign statement without unnecessary FMI calls; At the end, the FMU2MAST utility will sort all the variables into the groups in the order of connection points, continuous state variables and others. The reason of this order is to avoid renaming the names of connection point and system variable in MAST, thus keep the most important variable names unchanged from the original names in the FMU. The third step is to convert the XML tree into the MAST data tree. The new MAST tree categorizes the variables into six categories based on data types shown in figure 1: **inputs, outputs, constants, states, vals** and **vars**. With all the equivalent information available, FMU2MAST can generate a correct MAST wrapper model.

This new FMU2MAST utility has been verified by 44 cross-check FMUs exported from five different tools: FMUSDK, OpenModelica, MathWorks, Dymola and standard reference tests suggested by [3]. The generated MAST models have been tested in two different simulators: Saber and SaberHDL. The simulation results match well with the reference results provided by the examples. The tests are selected with the intention of covering as many different applications as possible: 1. Exported from five different tools; 2. Analog system: vanDerpol; 3. Discrete system: BooleanNetwork1; 4. Mixed-Mode system: BouncingBall; 5. Multi-domain system: for example, hydraulic ControlledTanks, mechanical CoupledClutches, electrical Rectifier and thermal ControlledTemp; 6. Coupling system with multiple FMUs: motor drive example; 7. Complex

| | vendor | input | output | discrete | state | others |
|-----------------|--------------|-------|--------|----------|-------|--------|
| Controlledtanks | OpenModelica | 0 | 0 | 246 | 4 | 535 |
| Motor_drive2 | Dymola | 0 | 1 | 39 | 15 | 253 |
| MixtureGases | Dymola | 0 | 2 | 4 | 16 | 76 |
| DFREG | Dymola | 0 | 2 | 40 | 0 | 44 |
| CoupledClutches | Dymola | 1 | 4 | 50 | 18 | 123 |
| Rectifier | Mworks | 0 | 8 | 6 | 4 | 177 |
| ControlledTemp | Mworks | 0 | 2 | 4 | 1 | 64 |
| BooleanNetwork | Mworks | 1 | 9 | 57 | 0 | 29 |
| FullRobot | Mworks | 0 | 6 | 109 | 36 | 5259 |
| BouncingBall | FMUSDK | 0 | 0 | 1 | 2 | 3 |
| vanderpol | FMUSDK | 0 | 0 | 0 | 2 | 3 |

Figure 6 cross-check tests information

system: 36 state variables, 109 discrete variables and more than 5000 other variables in FullRobot example. The detail information of some tests are listed in figure 6.

5 Bouncing Ball Example

A simplified MAST model of a bouncing ball example generated by FMU2MAST is given in the appendix. This example has two state variables. Although it is very simple, it can be used to illustrate two important simulation techniques. The first one is the accurate event time detection. It is challenging to detect the accurate event time in a mixed-mode simulator with a variable time-step integration algorithm. The second is the re-initialization of a state variable. It is trivial if the HDL provides this capability, such as **reinit()** in Modelica [5] and **break** in VHDL-AMS [6]. However, MAST language has no such a function. If these two features are not implemented appropriately in MAST wrapper model, the bouncing ball will not behave correctly. Three bouncing ball examples exported from different tools have been tested, only two of them succeed. The reason for the failed one is that the state variable velocity has no **<reinit>** attribute in its modelDescription.xml. After the **<reinit>** attribute is added manually to the velocity variable, it works as well as other two.

In the header of the MAST model attached in the appendix, the height h is defined as an output connect point, the gravitational constant g is defined as a model parameter with default value of 9.81. The elastic coefficient e is defined as an internal state variable with initial value of 0.7.

At the beginning of **parameters** section, **saberFMI initialization** interface is called to create the **fmiHandle** for the FMU model. Inside this interface, it will call standard **fmi2Instantiate** to instantiate the FMU. The **saberFMI setValues** interface is called to pass the gravitational constant g to FMU and update it by **fmi2SetReal**. After that, **saberFMI updateValues** is called, the FMI calculated parameters will be updated by standard FMI function **fmi2EnterInitialization** and **fmi2ExitInitialization**. At the end of **parameters** section, **saberFMI getValues** is called to get the initial values, h_ic is initial value for continuous state height h and v_ic is for velocity v . They will be used in the MAST **control_section** to set initial condition for the differential equations in operating point analysis.

The first **when** section with sensitive variables **dc_init** or **time_init** is called at beginning of operating point analysis and transient analysis. It processes all the initial events by FMI interface, **fmi2NewDiscreteStates**, until no new events are generated.

In **values** section, two **saberFMI nonlinear** interfaces are called to get FMU derivative variables: $der(v)$ and $der(h)$. Obviously, the equation $der(v) = -g$ has a constant relationship, and $der(h) = v$ has a linear relationship. Both equations don't have a nonlinear relationship but a nonlinear MAST function is used here. The reason is that the FMU model is a black box to the MAST model, there is no explicit expression for each equation, however, the linear/nonlinear relationship is available in the **ModelStructure** section in the model description file. According to FMI documentation, if the **<dependencies>** attribute is presented as an empty list, the *Unknown* depends on none of the *Knowns*; If the **<dependencies>** attribute is presented and **dependenciesKind** is constant or fix, then the *Unknown* has a linear relationship with the *Knowns*; If no **<dependencies>** attribute is provided, then *Unknown* has no particular dependency on *Knowns*. In this example, since both $der(v)$ and $der(h)$ have no **<dependencies>** attributes, they are translated as nonlinear dependency of state variables: v and h .

In order to detect the accurate events time, an interface **saberFMI checkCross** is introduced in the **values** section. It checks whether there is an event occurred in the time interval between the last accepted time and current time. If it is true, then **xIndicator** becomes 1, and 0 if it is false. In this example, the true means the ball hits ground, the height variable h becomes negative and the velocity variable needs to be re-initialized as $v = -e*v_0$. To achieve this, a MAST threshold detection section is used: **when (threshold (xIndicator, 0.5, before, after))**. This threshold function will use the simulator built-in threshold cross detection algorithm to find the exact time when the crossing event occurs. If the **xIndicator** crosses 0.5, a new event, **cross**, will be scheduled, which triggers another process **when (event_on (cross))** to force simulator to find a solution at this time point. With this build-in event detection method, the event time can be detected precisely with an

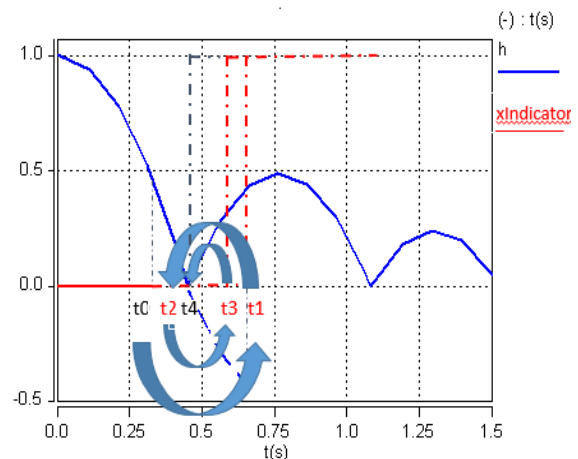


Figure 8 Threshold Crossing Detection

error less than one picosecond. The iterative event detect process can be shown in figure 8. Where t_0 is the last accepted time point, t_1 to t_3 are the tentative time points during the iteration, although the $xIndicator$ becomes 1, they are discarded because they don't meet the time criteria: $\Delta t \leq 1 ps$. t_4 is the final event time, it satisfies both event and time constraints: $xIndicator=1$ and $\Delta t \leq 1 ps$.

After the eventIndicator is found to become true, the voltage v needs to be re-initialized due to the attribute **reinit=true**. It is trivial for an HDL if it provides the re-initiation capability, such as Modelica [5] which has `reinit()` function to do it. However, there is no such function in MAST language. To be able to simulate this dynamic re-initialization behavior for a continuous state variable, an additional equation (2) is introduced. For example, $der(v) = -g$, where g is a constant, the v decreases linearly with time, a single MAST differential equation $d_by_dt(v) = -g$ cannot achieve this re-initialization behavior: $v = -e*v_0$. To initialize it dynamically, a new variable v_0 is introduced to represent the original differential equation, another discrete variable v_{init} is used to help describe the re-initialization behavior. The original state variable v with **reinit** attribute now have two equations: (1) and (2). Solve these two equations together can achieve the dynamic re-initialization in MAST wrapper model:

$$\frac{dv_0}{dt} = -g \quad (1)$$

$$v = v_{init} + \Delta v_0 \quad (2)$$

The v_{init} is the value re-initialized immediately after each event is detected, Δv_0 is the velocity difference between the solution of equation (1) at current time t and time when cross event was detected. The final solution v should be v_{init} plus Δv_0 . The solution for the state variable velocity with **reinit** attribute is shown in figure

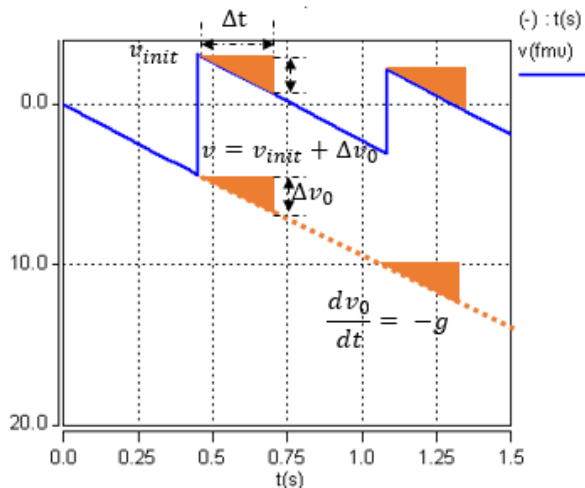


Figure 9 Solution for State Variable with reinit

9, the orange line represents the original differential equation (1) and the blue line represents the new equation (2) of a state variable with **reinit** attribute.

All these detail works of the event detection and state variable re-initialization are handled in FMU2MAST utility during the FMU conversion. Figure 10 is the Saber simulation results of the MAST model converted from the FMU: `ref_BRef.fmu` [3]. The transient analysis uses variable time-step integration algorithm. The initial time step is 1 us, then gradually increase to 100 ms around 0.4 second when the height of ball is close to zero. After the height of the ball becomes negative, the eventIndicator of the FMU becomes 1, the converted MAST model will find this event, re-initialize the state variable v_{init} at this time and save the continuous solution v_0 of equation (1). With v_0 and v_{init} the actual velocity can be calculated with equation (2). After this event, the time step is reset back to the initial time step of 1 us, and gradually increases based on truncation error until the ball hit the ground again.

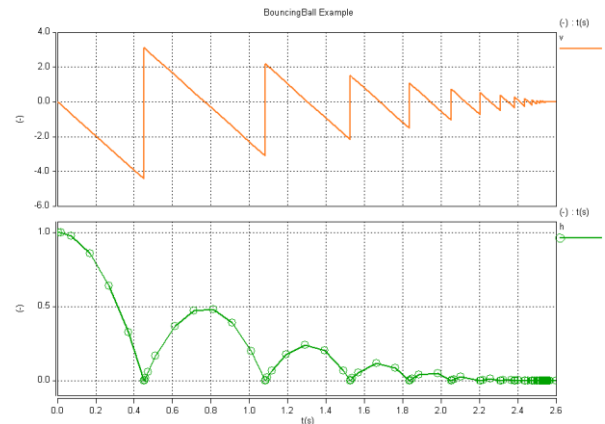


Figure 10 Bouncing Ball Results

Three bouncing ball examples have been tested. They are exported by FMUSDK, MathWork and [3]. Two of them worked well with this method but one failed. The reason for the failed one is that the velocity state variable doesn't have the attribute `<reinit>`, therefore, the translated MAST model doesn't know which variable needs to be re-initialized when the eventIndicator is detected. After manually adding attribute `<reinit=true>` to velocity in `modelDescription.xml`, the new generated MAST models works as well.

6 Motor Drive Example

When a system has multiple FMUs which are connected in a loop, the system needs to be solved by evaluating all the FMUs inside the loop repeatedly until the residue is close to zero [1]. This method works, but is difficult for a heterogeneous system that there are some non-

FMU models (MAST or VHDL-AMS) involved in the loop. To be able to handle more general applications, Saber simultaneously solves the DAE equations from all the models written in different languages. A typical DAE solver needs the partial derivatives of the equations to construct the Jacobian matrix during the iteration, and the FMI provides **fmi2GetDirectionalDerivative** interface for it. However, this interface is an optional in FMI standard, many FMUs don't provide it, except the FMUs exported by Dymola. It is known that one of the advantage of an HDL is that it does not require the modeler to provide the derivatives of the model equations. Saber simulator will analysis the MAST model, figure out the dependencies of all the linear/nonlinear equations, and approximate the nonlinear equations with PWL (Piece-Wise Linear) method [7]. With the PWL approximation, the partial derivatives can be obtained numerically without FMI interface **fmi2GetDirectionalDerivative**. This example will be used to discuss this method in Saber simulator.

This example is exported from Dymola. It has three FMUs: stimuli, controller and motor. The model connection diagram is shown in figure 11.

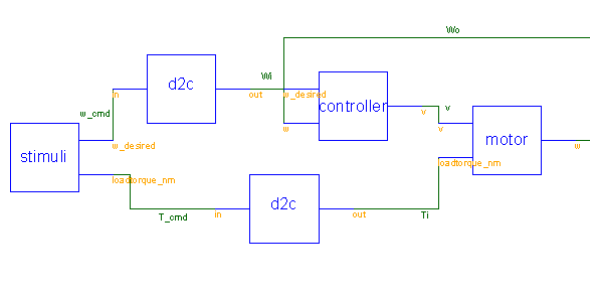


Figure 11 Motor Drive Schematic

The detail FMU implementations are embedded in binary code compiled from C source files. Even the FMU provides the original C files, it is still harder to understand the model characteristics when compared to an HDL. However, the model description file modelDescription.xml provides very helpful information, such as model inputs/outputs, state variables and dependency relationship of *Unknowns*. Based on this information, it is possible to derive the abstract equations of an FMU.

The first stimuli FMU sends out the reference angular speed and torque with respect to time. From its modelDescription.xml, the output equations can be derived as:

$$\begin{cases} \omega_i = u_1(t) \\ T_i = u_2(t) \end{cases} \quad (3)$$

$$(4)$$

Where ω_i is desired angular speed and T_i is the required torque.

The second controller FMU takes the desired angular speed ω_i from the stimuli, and the actual speed ω_o fed back from the motor as inputs, produces output voltage to the motor. From **<ModelStructure>** in the modelDescription.xml, it is known that the output angular speed depends on three variables with the same **dependenciesKind** of "fixed": the reference speed, the feedback speed and a state variable pi_x . The derivative $\frac{dpi_x}{dt}$ depends on both the reference speed and feedback speed with the same **dependenciesKind** of "fixed". According to the FMI documentation, the "fixed" **dependenciesKind** represents the *Unknown* depends on a *Known* with a fixed factor, and the factor is an expression that is evaluated before the **fmi2ExitInitializationMode** is called. Based on this, the characteristic equations of the controller model can be derived as:

$$\begin{cases} \frac{dpi_x}{dt} = k_1 * \omega_i + k_2 * \omega_o \\ v = k_3 * \omega_i + k_4 * \omega_o + k_5 * pi_x \end{cases} \quad (5)$$

Where pi_x is a state variable, v is output voltage for the motor, ω_i is reference angular speed from the stimuli, ω_o is the actual angular speed fed back from the motor, k_1 to k_5 are fixed coefficients. If $k_1 = -1$, $k_2 = 1$ and $k_3 = -1$, $k_4 = 1$, then the model is a classic PI (Proportional-Integral) controller, and the state variable pi_x is the integral of the speed error.

The third FMU, the motor model, takes the controller output voltage and stimuli torque command as inputs, delivers required torque and maintains the desired motor speed. From its **<ModelStructure>** in the modelDescription.xml, it is known that this FMU has one output variable and three continuous state variables, the characteristic equations of motor can be written as:

$$\frac{di}{dt} = f_1(v, i, \omega) \quad (7)$$

$$\frac{d\omega}{dt} = k_6 * i + k_7 * T_i \quad (8)$$

$$\frac{d\theta}{dt} = \omega \quad (9)$$

$$\omega_o = k_8 * \omega \quad (10)$$

Where i is the motor current, v is the voltage applied on the motor, ω is the internal speed with unit of *rad/ps*, θ is the rotation angle which is the integral of angular speed, ω_o is the motor output speed, k_6 to k_8 are fixed coefficients. The motor current derivative $\frac{di}{dt}$ depends on the motor voltage v , current i and the motor angular

speed ω with the same **dependenciesKind** of “dependent”. According to the FMI standard, if the **dependenciesKind** is “dependent”, it means the *Unknown* depends on the *Known* without a particular structure. This “dependent” **dependenciesKind** is treated as a nonlinear dependency during conversion from an FMU to a MAST model.

There are two extra blocks, d2c, used in the design. They are hyper-models to convert the discrete output to continuous output. The reason for this is that the angular speed and torque outputs of stimuli are discrete outputs while the inputs of the controller and the motor are continuous inputs. It is illegal in MAST language to connect the different type of ports together. With these two hyper-models, it is possible to drive the continuous ports of the controller and the motor with discrete speed and torque output.

When all the three models are connected in a loop, there are 8 equations in total. Equations (3), (4), (6) and (10) are algebraic equations, while (5), (7), (8) and (9) are differential equations. When the design is loaded into Saber simulator, the simulator will setup these equations into the following DAE form [4][8]:

$$Ax + E\dot{x} = B(t) \quad (11)$$

Where

$$x = \begin{pmatrix} \omega_i \\ T_i \\ p_i \\ v \\ i \\ \omega \\ \theta \\ \omega_o \end{pmatrix}, \quad \dot{x} = \begin{pmatrix} \dot{\omega}_i \\ \dot{T}_i \\ \dot{p}_i \\ \dot{v} \\ \dot{i} \\ \dot{\omega} \\ \dot{\theta} \\ \dot{\omega}_o \end{pmatrix}$$

the matrix E is:

$$E = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

It is a constant sparse matrix. It is also structural singular due to the diagonal value in the row 1, 2, 4 and 8 are zeros, then the equation (11) is a DAE system.

As stated earlier, equation (7) is a nonlinear equation and it needs to be linearized in an iterative form with Newton-Raphson method to be solved:

$$F'(X_{k-1})X_k = -F(X_{k-1}) + X_{k-1}F'(X_{k-1})$$

With the Newton-Raphson method, the Jacobin matrix A of equation (11) in the nonlinear iteration is:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -k_1 & 0 & 0 & 0 & 0 & 0 & 0 & -k_2 \\ -k_3 & 0 & -k_5 & 1 & 0 & 0 & 0 & -k_4 \\ 0 & 0 & -\frac{\partial f_1}{\partial v} & -\frac{\partial f_1}{\partial i} & 0 & -\frac{\partial f_1}{\partial \omega} & 0 & 0 \\ 0 & -k_7 & 0 & 0 & -k_6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -k_3 & 0 & 1 \end{pmatrix}$$

Also the iterative RHS (right hand side) $B(t)$ of the equation (11) becomes:

$$\begin{pmatrix} u_1(t) \\ u_2(t) \\ 0 \\ 0 \\ -f_1(v_{k-1}, i_{k-1}, \omega_{k-1}) + \frac{\partial f_1}{\partial v} v_{k-1} + \frac{\partial f_1}{\partial i} i_{k-1} + \frac{\partial f_1}{\partial \omega} \omega_{k-1} \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

All the partial derivatives in Jacobin matrix A and B are unknown, and need to be calculated in the iteration. FMI standard provides **fmi2GetDirectionalDerivative** to get partial derivatives, but it is optional in FMI standard, and not available in all FMUs. A simple numerical differentiation method is used in Saber to calculate the derivatives.

$$f'(x) = \lim_{h \rightarrow 0} \left(\frac{f(x+h) - f(x)}{h} \right) \quad (12)$$

In matrix A , all the coefficients k_1 to k_8 are fixed and won't change after **fmi2ExitInitializationMode** is called. These constant coefficients are calculated in MAST parameter section when the design is loaded into simulator. The coefficients in 5th row of matrix A are the partial derivatives of the nonlinear equation (7), and varying during the nonlinear iteration. These coefficients are calculated in MAST values section with nonlinear PWL approximation. In the PWL approximation method, each dimension of a nonlinear function is divided into many small subdivisions, named **sample_points** [7], and in each subdivision, the nonlinear function is approximated with a linear function. The smaller the subdivision width h , the more accurate the nonlinear approximation is. The subdivision width h is controlled by **sample_points** specification, it can be adjusted by user when there is a need for better accuracy. All the MAST equations

translated from an FMU by FMU2MAST utility will maintain the same linear/nonlinear relationship in the FMU. With the PWL method, the DAE equations (11) can be solved simultaneously in Saber simulator.

Figure 12 are the transient analysis results of the motor drive example with a variable time-step integration method. The initial time step is 1 us. The reference angular speed ω_i is set to 10 rpm at time 0.1 second. The actual motor speed reaches 10 rpm around 0.2 second, about 0.1 second delay from the reference. After another 0.1 second it settles down 10 rpm at 0.3 second. At 0.5 second, the stimuli model applies 3 Nms torque on the motor shaft, the voltage required to maintain the speed is reduced, as shown with the purple curve, the voltage drops from 6.5 volts to 6 volts after 0.6 second. In this example, the variable time-step integration method is used in transient simulation. As it is shown in figure 12, the marks on the purple curve v indicate the exact simulation time points during the simulation. Whenever a step change occurs on the input signals, the truncation error (LTE) of the differential equations increases. To control the accuracy of results, the simulator automatically reduces the time step to make the LTE less than simulator setting. The smallest time step is about 50 nanoseconds, 1/20 of the initial time step (1 us). After the motor reaches its steady-state speed, 10 rpm, the LTE becomes so small, the simulator automatically increases the time step to improve the simulation speed without sacrificing the accuracy. As it is shown, at 0.9 second, the time step has been increased up to 0.1 second, which is about 100000 times of the

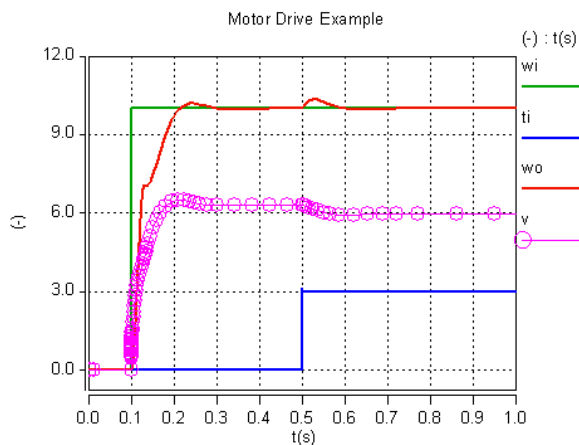


Figure 12 Motor Drive Results

initial time step (1 us). Compared to the fixed time step algorithm, the variable time-step integration algorithm significantly improves the simulation speed.

7 Conclusion and Future Work

In this paper, a new method of importing an FMU to a Hardware Description Language is introduced. A conversion utility, FMU2MAST, is developed to help

the conversion from FMUs to MAST models automatically. This method has been proved a success with 44 FMUs exported from five different tools. With this method, the FMUs can be imported to another simulator which supports MAST language without any extra work. The converted MAST wrapper model can also be simulated with other non-FMU models which written in MAST or VHDL-AMS language to help the study of a more complex heterogeneous system. The FMI 2.0, the version used in the paper, doesn't support the general DAE system yet, however, this can be easily extended to support it when the FMI standard supports the DAE in the future. Right now this method is only applied to the FMI 2.0 for Model Exchange, but it can also be applied to support the FMU import for Co-Simulation as well in the future.

8 Appendix

Attached is the simplified MAST code generated from ref_bBRef.fmu. To make the model meaningful for illustration, only the relevant codes are kept.

References

1. Functional Mock-up Interface for Model Exchange and Co-Simulation, 2.0 July 25, 2014. <https://www.fmi-standard.org/downloads>
2. OpenMAST Language Reference Manual, 1.0, June 2004.
3. Christian Bertsch, Award Mukbil, Andreas Junghanns, Improve Interoperability of FMI-supporting Tools with Reference FMUs, pp. 533-540, Proceedings of the 12th International Modelica Conference, May 15-17, 2017, Prague, Czech Republic
4. R. Scott Cooper, The Designer's Guide to Analog & Mixed-Signal Modeling, March 1, 2001
5. Modelica – A Unified Object-Oriented Language for Physical System Modeling Language Specification, Version 3.0, September 5, 2007
6. Peter J. Ashenden, Gregory D. Peterson, Darrel A. Teegarden, The System Designer's Guide to VHDL-AMS: Analog, Mixed-Signal, and Mixed-Technology Modeling, September 10, 2002
7. Saber Simulator Guide: Reference Manual, Synopsys, June 2006
8. Yutaka Hirano, Satoshi Shimada, "Initiatives for acausal model connection using FMI in JSAE", Proceedings of the 11th International Modelica Conference September 21-23, 2015, Versailles, France
9. T. Blochwitz, M. Otter *et al.*, "Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models", Proceedings of the 9th International Modelica Conference, September 3-5, 2012, Munich, Germany

```

# THIS MODEL IS NOT A COMPLETE MODEL. ONLY FOR DEMONSTRATION!
element template fmu_bbref h = g
number g=9.81 # acceleration of gravity.
{
  # variable declarations...
  parameters {
    fmiHandle = saberfmi(initialization,fmiHandle,instance(), "fmu_bBRef.fmu")
    constErr = saberfmi(setValues,fmiHandle,-1,g,g_id,fmiReal)
    constErr = saberfmi(updateValues,fmiHandle,g)
    h_ic = saberfmi(getValues,fmiHandle,h_id,fmiReal)
    v_ic = saberfmi(getValues,fmiHandle,v_id,fmiReal)
  }
  when(dc_init|time_init) {
    stateErr = saberfmi(initEvents,fmiHandle)
    v_init = v_ic
  }
  when(threshold(xIndicators,0.5,before,after) & after >0 ) {
    schedule_event(time,cross,1.0)
  }
  when(event_on(cross) & time_domain) {
    stateErr = saberfmi(updateCross,fmiHandle,time,h,h_id,v,v_id)
    nextEvent = saberfmi(checkEvents,fmiHandle,time)
    hasBreak = saberfmi(valuesChanged,fmiHandle)
    v_init = saberfmi(timeValues,fmiHandle,time,v_id,fmiReal)
    prev_v_0 = v_0
    schedule_next_time(time)
    e = saberfmi(getValues,fmiHandle,e_id,fmiReal)
  }
  when(dc_done|time_step_done) {
    stateErr = saberfmi(acceptValues,fmiHandle,h,h_id,fmiReal,v,v_id,fmiReal)
    e = saberfmi(getValues,fmiHandle,e_id,fmiReal)
    nextEvent = saberfmi(stepDone,fmiHandle,time)
    schedule_next_time(nextEvent)
  }
  values {
    der_h = saberfmi(nonlinear,fmiHandle,time,der_h_id,h,h_id,v,v_id)
    der_v = saberfmi(nonlinear,fmiHandle,time,der_v_id,h,h_id,v,v_id)
    xIndicators = saberfmi(checkCross,fmiHandle,time,h,h_id,v,v_id)
    delta_v_0 = v_0 - prev_v_0
  }
  control_section {
    initial_condition(h,h_ic)
    initial_condition(v_0,v_ic)
  }
  equations {
    h : d_by_dt(h) = der_h
    v_0 : d_by_dt(v_0) = der_v
    v : v = v_init + delta_v_0
  }
}

```

Developing a Framework for Modeling Underwater Vehicles in Modelica

Shashank Swaminathan¹ Srikanth Saripalli¹

¹Texas A&M University, College Station, TX, USA, sh.swami235@gmail.com, ssaripalli@tamu.edu

Abstract

When developing Remotely Operated Vehicles (ROVs), models prove extremely useful in determining design parameters and control strategies. This paper's goal is to develop a modeling framework for underwater ROVs in Modelica, with integration with the Robotic Operating System (ROS), allowing for quicker prototyping and testing of ROV design and control.

Named the Underwater Rigid Body Library (URBL), the modeling framework treats the effect of water on submerged bodies as interactions with a "field" of water to capture the effects of buoyancy and drag. Its usage is demonstrated by applying it to the BlueROV2, a commercially available ROV from Blue Robotics. Using controller signals to the propellers as system inputs, the model was tested with various motor command profiles to achieve different composite motions. Constant motor commands were provided both from within Modelica and from ROS; the simulation results indicated that the model responded appropriately.

Keywords: Underwater, ROV, Modelica, ROS, Framework

1 Introduction

1.1 Relevant Background

Remotely Operated Vehicles (ROVs) are vital for the exploration and development of areas that are beyond the reach of humans, particularly in the underwater field. When developing any ROV design, it is helpful to construct a model of the design to provide an idea of its performance. There exist many models of underwater vehicle designs like in (Prester, 2001), but these models focus specifically upon one vehicle design. There are very few initiatives geared towards modeling a variety of underwater bodies and vehicles (McMillian *et al.*, 1995; Tran *et al.*, 2018), but even these are purpose-built software programs. The aim of this paper is to develop a general-purpose modeling framework in Modelica that can be used to model an ROV design using prebuilt components and has flexibility to grow as a library.

The paper will focus on modeling the ROV based off rigid-body principles, as is done in (Tang, 1999; Wang,

W. *et al.*, 2006). This is as opposed to modeling based on CFD principles, like in (Yang *et al.*, 2016; Wang, C., *et al.*, 2014), as it would be intractable for quick prototyping and control testing. Representing hydrodynamic forces, such as viscous drag and added mass, can be done at varying levels of complexity, as seen in (Yuh, 1990), and (da Silva *et al.*, 2007). As this paper's focus is on developing a modeling framework, it will only address the most basic of hydrodynamics, while also providing a template for further expansion by the user.

1.2 Objectives

The goal for the work described in this paper is to develop a basic framework for mathematically modeling underwater vehicles that can:

- Aid the prototyping and testing of vehicle design and controls.
- Be readily integrated with common control and feedback mechanisms, specifically ROS.
- Visualize prototype design and test results via three-dimensional animation.

In Section 2, the modeling framework URBL is discussed in detail. In Section 3, a demonstration of the modeling framework is done via a use case of modeling a physical ROV. Section 4 follows with verification tests of the model developed from the framework, and the ROS interface capability of the model. Section 5 provides the final remarks and closes the paper.

2 Underwater Rigid Body Library.

2.1 Overview

The modeling framework is developed as the Underwater Rigid Body Library (URBL) – the library contains the base functional components to any ROV design. The URBL consists of two major sections – components and interfaces for modeling underwater vehicles, and an external interface to ROS. The URBL components are models to describe rigid body interactions with water; the interface is for a basic propeller.

2.2 Rigid Body and Field Model

As modeling within Modelica is component-based, it is imperative to develop a rigid body component that can interact with the surrounding submerging fluid environment. To accomplish this, the framework represents the environment via a *field*. The fluid is assumed to be an incompressible Newtonian fluid. The field is considered to have two primary interactions with the rigid body – one via the buoyancy and the other via fluid drag forces exerted by the fluid on the rigid body. This is written into the model as shown below, with ρ_{body} being the density of the rigid body, ρ_{fluid} the density of the submergent fluid, $\nu_{viscosity}$ being the coefficient of viscosity, and A being the cross-sectional area of the body.

$$\vec{f}_{buoyant} := -\frac{\rho_{fluid}}{\rho_{body}} \cdot m \cdot \vec{g} \quad (1)$$

$$\vec{f}_{drag} := -\nu_{viscosity} \cdot A \cdot \vec{v} \quad (2)$$

Here, \vec{v} represents the velocity of the body, \vec{g} the gravitational acceleration, and m the mass. To account for drag torques purely due to angular speed $\vec{\omega}$, the following equation is added to the drag computation:

$$\vec{\tau}_{drag} = -k_{drag} \cdot \vec{\omega} \quad (3)$$

where k_{drag} represents the coefficient of drag rotationally (Wadoo, Kachroo, 2016).

The field model dictates the values of the forces affecting bodies within. The field's force is applied equally across all elements in the field. However, when dealing with a rigid body, where the only interface available is the Frame of Interest (F.O.I), it is not possible to implement the field in such a manner. Instead, the total force the field applies on the body at the center of mass (CM) is translated to the F.O.I, as seen in Figure 1.

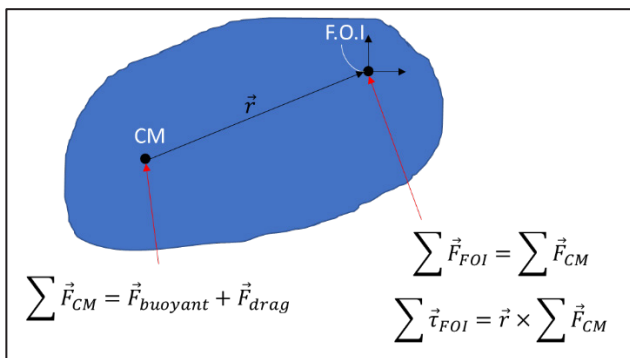


Figure 1 – Translation of buoyant and drag forces from center of mass to frame of interest

The rigid body model itself is extended from the standard MultiBody Library (Otter, 2003). The field model is added to the rigid body model, using the **inner** and **outer** qualifiers in Modelica, so that any component constructed from this rigid body model will interact with

the fluid surrounding the component, regardless of design. The Modelica-specific implementation is shown in Appendix B.

2.3 Propeller Model

The schematic in Figure 2 captures the torque and thrust generation in the propeller – the electric motor is captured through the EMF, and the propeller frame captures the momentum exchange between the blades and the water.

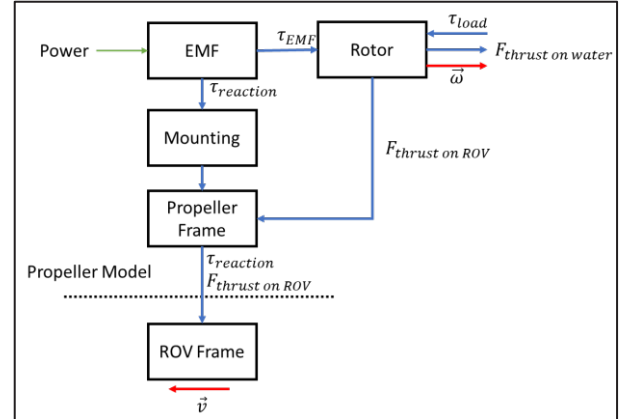


Figure 2 - Schematic of propeller structure

The propeller's rotor is powered by a motor, which in turn is powered by some external source of power. From (Triantafyllou, 2004), the thrust can be written as proportional to the square of the rotor's angular velocity ω .

$$F_{thrust} \propto \omega^2, K_T(J^*) \quad (4)$$

$K_T(J^*)$ is the thrust coefficient, where J^* is the ratio between rotor speed (intake speed) and fluid speed (outtake speed). Specifically, $K_T(J^*)$ can be approximated (Triantafyllou, 2004) as follows:

$$K_T(J^*) = \beta_1 - \beta_2 J^* \quad (5)$$

$$J^* = \frac{v}{\omega} \quad (6)$$

Here, β_1 and β_2 are functions of the intake and outtake speeds of the water.

Taking v and ω as the linear and angular velocities of the propeller along its axis, Equation 4 can be rewritten as follows:

$$F_{thrust} \propto \omega^2 \left(\beta_1 - \beta_2 \frac{v}{\omega} \right) \quad (7)$$

Letting k_r, k_m be appropriate constants of proportionality, it can be rewritten as

$$\vec{F}_{thrust} = k_m |\omega| (k_r \vec{\omega} - \hat{\omega} \cdot \vec{v}) b_{dir} \quad (8)$$

where b_{dir} is a constant that indicates the direction of the propeller's mounting.

While the load torque on the propeller due to thrust can also be represented similarly, for the sake of simplicity, it is approximated by a power balance with constant efficiency η , as seen in equation 9. An additional $-k_{loss} \vec{\omega}$ term is added to represent loss purely due to rotor rotation.

$$\vec{\tau}_{load} = -\frac{\vec{F}_{thrust} \hat{\omega} \cdot \vec{v}}{|\hat{\omega}| \eta} - k_{loss} \vec{\omega} \quad (9)$$

To better handle when ω approaches zero, the \vec{F}_{thrust} is expanded to rewrite the load torque as

$$\vec{\tau}_{load} = -k_m \vec{v} (k_r \vec{\omega} - \hat{\omega} \cdot \vec{v}) b_{dir} - k_{loss} \vec{\omega} \quad (10)$$

The hydrodynamic effects of added mass and wave drag are not considered in this implementation.

The component diagram implementation in Modelica is displayed in Figure 3. The propeller is split into two sections: the mass of the housing, represented by a URBL body, and the actual propeller rotor, represented by a *RotorID* component. The propeller is driven by an EMF; the *MountingID* components is used to propagate the load torques from the propeller to the main ROV body. The thrust is calculated as a *WorldForce* component and is applied to the mass of the propeller's housing directly; the load torque from the water is applied to the rotor as a one-dimensional torque, leaving it uncoupled from the actual ROV.

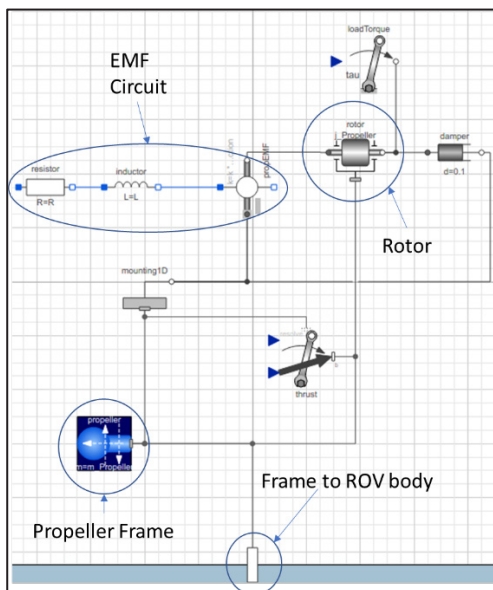


Figure 3 - Implementation of propeller in Modelica

2.4 Integration of External Controllers

Apart from providing the foundational components for modeling ROVs, the URBL's goal is also to provide easy integration with the Robot Operating System (ROS). ROS (Quigley, 2009) based controllers primarily rely on TCP/IP connections for communication. The URBL thus includes integration for socket communication to ROS, achieved via Modelica's external C function capability.

The integration is done via a block extended from a Multiple-Input-Multiple-Output (MIMO) block from the Modelica Standard Library. The extended block calls upon an external C function based on a time sampler function; the C function returns an array of control values read from the incoming information

queue buffer on the socket port. The socket uses TCP protocol for communication, allowing for explicit ordering of the flow of information – as opposed to UDP protocol. The block contains parameters to set the IP and port of the external controller. To have ROS interact with the model, a ROS node running a TCP socket was also written, allowing the ROS architecture to communicate with the model by using the node-socket connection as a relay point. The flow of data is shown in Figure 4.

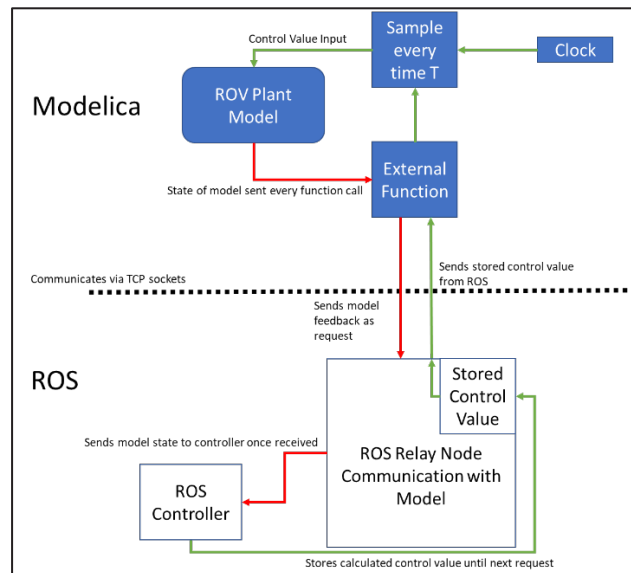


Figure 4 - Schematic of Data Flow between Modelica and ROS

2.5 Package Structure

Figure 5 shows the package structure of the URBL.

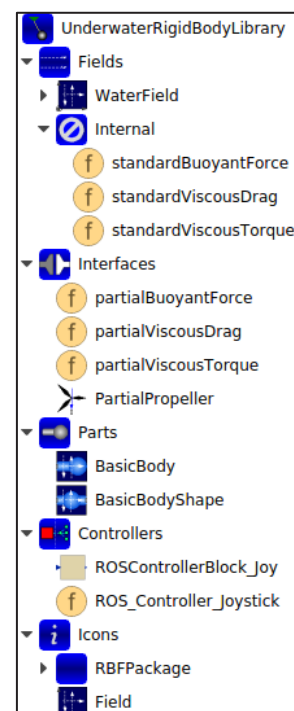


Figure 5 - Package Structure of the URBL

2.6 Development Review

The construction of the modeling framework was done in Ubuntu 16.04 using Wolfram SystemModeler (SystemModeler, Wolfram). The distribution of ROS used for testing integration capabilities with control platforms was ROS Kinetic. As the mechanism for connecting ROS to Modelica was based on using TCP sockets, and the build of the model was done in a Linux environment, the ROS connectivity is currently only usable in **nix* environments.

3 Application of URBL

The URBL's applicability is tested by modeling a commercially available ROV design – the BlueROV2 (BlueROV2, Blue Robotics), shown in Figure 6.



Figure 6 - Physical BlueROV2

The BlueROV2 has 6 propellers mounted – 2 dual vertical thrusters, and 4 vector-configured thrusters, allowing for 6 DOF. It is controlled via a Pixhawk Autopilot flight controller running ArduSub. The full hardware breakdown of the ROV is shown in Figure 7.

3.1 Frame Modeling

The process of assembling the frame of the BlueROV2 physically from kit is replicated when developing the model of its frame. The ROV is built from a base plate, two side plates, and four top plates, each a rigid body of certain uniform density and mass, with points on the body to connect with other parts of the frame. Likewise, the frame model was constructed from several sub-components, each representing one type of frame plate – bottom, side, and top – constructed from URBL rigid bodies, with frames to represent attachment points to other bodies. The resultant total frame is shown in Figure 8.

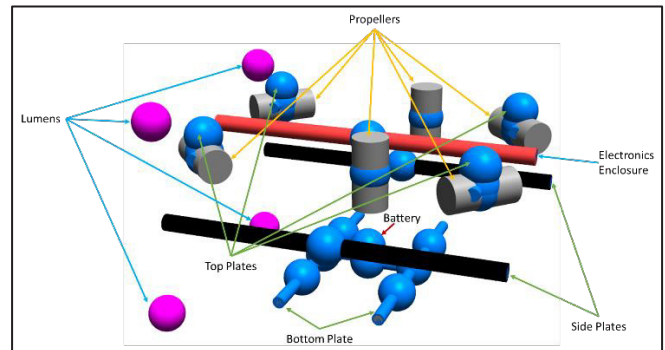


Figure 8 - Visualization of the ROV model

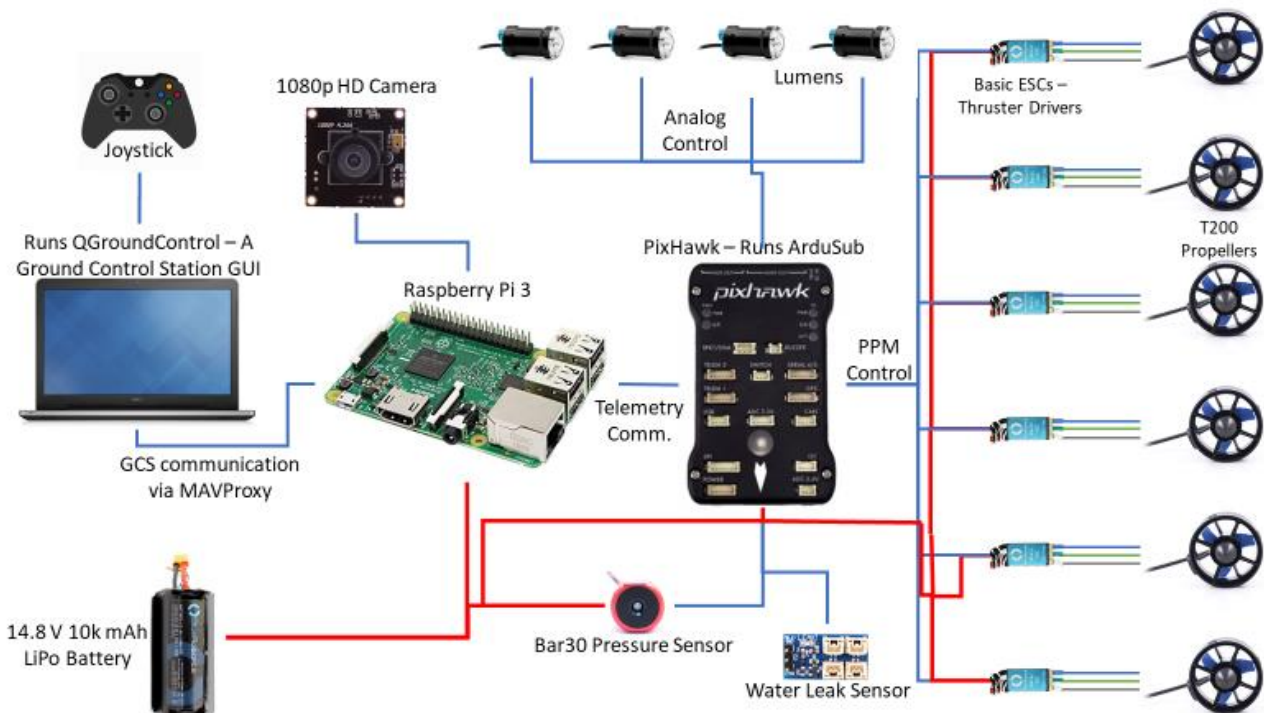


Figure 7 - Hardware schematic of power and information flow

3.2 Propeller Modeling

The propeller component for the BlueROV2 is extended from the URBL's base propeller model. The T200 propellers, used on the BlueROV2, are controlled via Pulse Position Modulation – to approximate this voltage control, a standard signal voltage component was used. Each propeller thus has its own internal electric circuit, with the signal voltage value controlled externally. By doing so, it allows for simpler testing against flight data from the physical ROV – the Pixhawk flight controller on the BlueROV2 sends pulses to the propeller's driver ESC, which then controls the voltage to the propeller. Hence, the model can now run the same commands sent by the Pixhawk and ESC driver to the propeller.

The fore-aft propellers are all oriented at 45-degree angles, for lateral movement, while the vertical propellers are mounted perpendicular to the mounting plate – as shown in Figure 9. Note that propellers 1 and 2 are facing forward, while propellers 3 and 4 are facing backwards; propeller 6 is upwards facing, while propeller 5 is downwards facing.

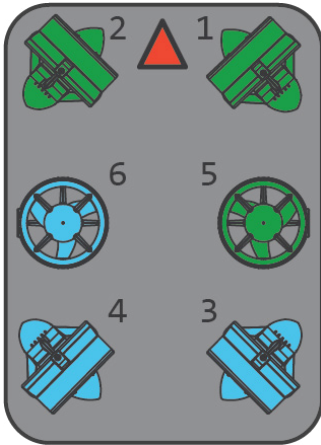


Figure 9 - Propeller orientation diagram [10]

3.3 Integration of ROS

The integration with ROS from the URBL library was used to receive control values for propeller actuation. A joystick was used to provide the values for composite motion – to translate these to control values per each propeller, a separate controller node was created – the flow of control input is shown in Figure 10.

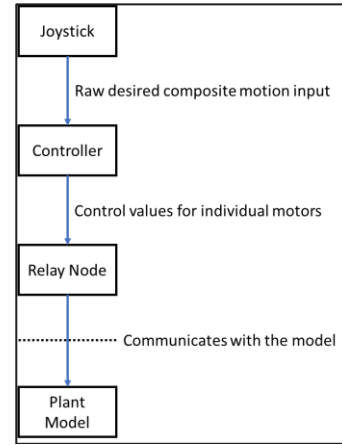


Figure 10 - ROS-based control input flow

The relationship between the six propeller torques and the resultant forces and torques along three dimensions was derived as follows:

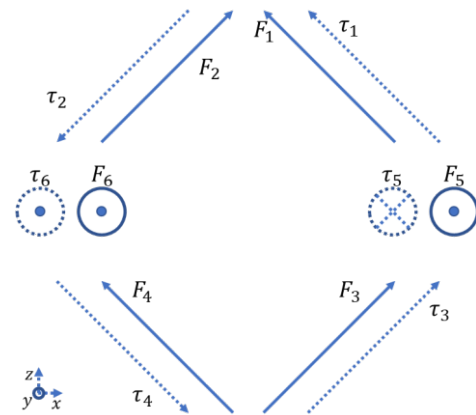


Figure 11 - Propeller force and torque orientation

Following the orientation of the propellers shown in Figure 11, the equations of forces and torques generated by each propeller were derived: $\vec{\tau}_i$ represents the reaction torque generated by the propeller, \vec{F}_i the force acting on the propeller's center of mass, \vec{h}_i the vector from the propeller's center of mass to that of the ROV, and τ_{F_i} the reaction torque acting on the ROV due to thrust.

$$\vec{\tau}_1 = \tau_1 \sin 45 \hat{i} - \tau_1 \cos 45 \hat{k} \quad (11)$$

$$\vec{F}_1 = F_1 \sin 45 \hat{i} - F_1 \cos 45 \hat{k} \quad (12)$$

$$\begin{aligned} &= n\tau_1 \sin 45 \hat{i} - n\tau_1 \cos 45 \hat{k} \\ \vec{\tau}_{F_1} &= \vec{h}_1 \times \vec{F}_1 = (h_{1x}\hat{i} + h_{1y}\hat{j} + h_{1z}\hat{k}) \times \vec{F}_1 \end{aligned} \quad (13)$$

The relationship between propeller torque and propeller thrust is approximated as proportional for the purposes of deriving a basic control matrix. The torque and force relationships for the other propellers are similar to Equations 11 through 13 above, with differences in orientation. This leads to the invertible matrix shown in the left of the equation in Figure 12,

$$\begin{pmatrix} \frac{n}{\sqrt{2}} & \frac{n}{\sqrt{2}} & \frac{n}{\sqrt{2}} & \frac{n}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & n & n \\ -\frac{n}{\sqrt{2}} & \frac{n}{\sqrt{2}} & \frac{n}{\sqrt{2}} & -\frac{n}{\sqrt{2}} & 0 & 0 \\ \frac{1-nh_{1,y}}{\sqrt{2}} & \frac{nh_{2,y}-1}{\sqrt{2}} & \frac{nh_{3,y}+1}{\sqrt{2}} & \frac{-nh_{4,y}-1}{\sqrt{2}} & -nh_{5,z} & -nh_{6,z} \\ \frac{n(h_{1,x}+h_{1,z})}{\sqrt{2}} & \frac{-n(h_{2,x}-h_{2,z})}{\sqrt{2}} & \frac{-n(h_{3,x}-h_{3,z})}{\sqrt{2}} & \frac{n(h_{4,x}+h_{4,z})}{\sqrt{2}} & -1 & 1 \\ \frac{-nh_{1,y}-1}{\sqrt{2}} & \frac{-nh_{2,y}-1}{\sqrt{2}} & \frac{1-nh_{3,y}}{\sqrt{2}} & \frac{1-nh_{4,y}}{\sqrt{2}} & nh_{5,x} & nh_{6,x} \end{pmatrix} \begin{pmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \\ \tau_4 \\ \tau_5 \\ \tau_6 \end{pmatrix} = \begin{pmatrix} F_i \\ F_j \\ F_k \\ \tau_i \\ \tau_j \\ \tau_k \end{pmatrix}$$

Figure 12 - Relationship between motor torques and composite motion

describing the relationship between propeller torques and composite motion. By applying the inverse of this matrix to scale the joystick input, the control values were derived.

3.4 Full ROV Model

The full ROV model is created by adding the propeller to the frame model, to provide the methods of propulsion and control to the ROV structure. Selected parameterization of the model is listed in the Appendix A. The completed ROV model is shown in Figure 13.

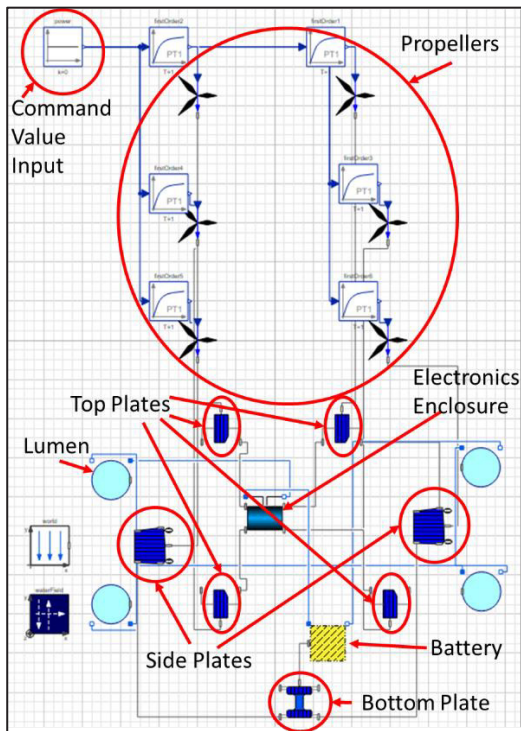


Figure 13 - Component view of full ROV model in Modelica

4 Testing the ROV Model

4.1 Component Testing

The purpose of the component tests is to verify that the component's individual performance conforms to expectations.

4.1.1 Frame Model Tests

When testing the frame, the frame sub-components are placed alone in a body of water, and their size, structure, and motion in response to buoyancy is verified – as each sub-component of the frame is constructed from HDPE (density of 0.97 g/cm³) and symmetric, it has a net buoyancy of 0.2 kg, and therefore is expected to slightly float upwards. The test results do indicate that all sub-components, along with the entire frame, exhibit normal, stable motion in the water field.

4.1.2 Propeller Model Tests

This test checks the propeller's ability to provide thrust to a rigid body in water. To check the model's stability during rotation, the propeller is made to provide thrust along different axes of rotation to the end of a neutrally buoyant rod. The test results indicate that the propeller proceeds stably and smoothly in all orientations, matching the expected motion.

4.2 Full Model Testing

The full ROV model is tested by providing a constant joystick command and evaluating the resulting composite motion of the ROV. The tested composite motion is the forward motion along the X axis – the control values necessary are derived from inverting the matrix in Equation 14.

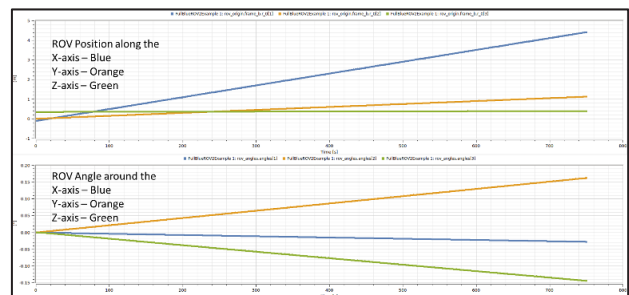


Figure 14 - Results from testing the full ROV model

In Figure 14, the motion along the X axis is stable, while the motion along the other axes after accounting for drift, linear and angular, is near zero. The drift seen in the rotational values can be attributed to approximations made when constructing the control matrix. The movement seen along the Y axis is due to the net buoyancy of the ROV, and therefore acceptable.

4.3 Testing ROS Integration

To test the validity of the model's external control capability – its connection to ROS – a network of ROS nodes meant to handle both the model's feedback and the provision of control values is setup. A joystick is used to dictate simple motor control values to the model, via ROS, and the model's reaction to the values is observed. The joystick sends simple motion commands in the orthogonal directions – lateral motion in the XZ plane – the raw joystick input is seen in Figure 15. The model's response is displayed in Figure 16. Note that the Figure 15 was recorded from ROS and uses the operating system time; this is different from the simulation time seen in Figure 16.

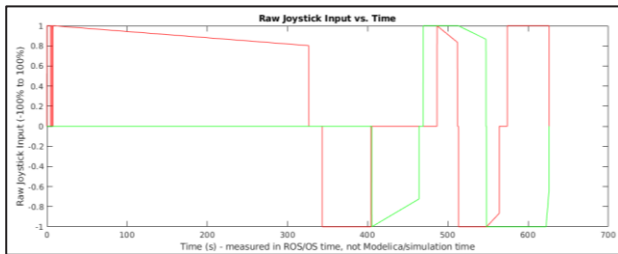


Figure 15 - Raw Joystick Input

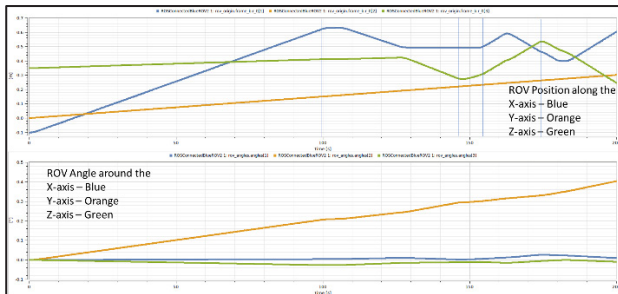


Figure 16 - Results from testing the ROV model when providing motor commands from ROS

The motion along the X axis, and the motion along the Z axis are the values being controlled by the controller. Drift in the rotation angles around the axes is seen, attributable to approximations done in the control matrix. A steady, slow rise is seen along the Y axis (in orange) due to the net buoyancy of the model. The ROV is controlled to move along the XZ plane in accordance with the joystick input. The response of the ROV is as desired, with appropriate motion when moving along each axis separately, as well as when moving in a composite manner in the XZ plane.

5 Conclusions

5.1 Results Summary

- The URBL was stably constructed to provide basic ROV modeling components, as well as ready-to-use integration with ROS

- The URBL was successfully used to model an existing commercially available ROV design, the BlueROV2.

5.2 Further Work

5.2.1 Library Improvements

The model of damping was simplified to take the cross-sectional area of a given component in the plane perpendicular to motion as a parameter – an improvement would be to have this area as a changing quantity.

The library's hydrodynamic models are overall extremely simplified, and so are currently implemented via functions, to increase replaceability. However, this possible interchangeability of hydrodynamic force functions is still limited in scope by the function interface; it could be widened to accept and return any number and kind of inputs and outputs.

For integration with external control mechanisms, the current socket-based integration relies on using Modelica's external C function capability and poses restrictions on the operating system used for simulation – **nix* based distributions, and not Windows. Socket based communication also has limitations in speed – the larger and more computationally intensive the model, the slower the socket-based communication will be. Further improvement can be done by porting this integration to rely on FMI/FMU functionality, instead of C functions and sockets. As noted by a reviewer, there exists another library for providing TCP/IP connections from Modelica via external C-functions, named the *Modelica_DeviceDrivers* library (Thiele, 2017). The ROS integration in this paper was developed separately from *Modelica_DeviceDrivers*, though both rely on TCP/IP communications.

5.2.2 Model Improvements

When prototyping the design of the model, it is useful to individually model the bodies involved in the ROV structure. However, this adds complexity to the model, and makes it simulate slower. Per a reviewer's suggestion, to speed up simulation post prototyping, the model should be redrawn with all the rigid bodies consolidated into one central mass, to improve simulation usefulness.

5.2.3 Validation Improvements

The motion profiles tested in the standalone model tests could be increased in complexity, from simple movements across and around axes, to more composite motion in three dimensions. The simulation results should also be compared against experimental data from the physical vehicle.

Appendix A – Physical Parameters

This is the list of the derived parameters for the electronics enclosure and the battery enclosure.

| "Parameter Name" | "Mass (kg)" | "Density (g/cm3)" |
|--------------------------------|-------------|-------------------|
| "Ballast" | "0.2" | "11.34" |
| "Battery Enclosure w/ Battery" | "1.654" | "0.83" |
| "Bottom Plate" | "0.25" | "0.97" |
| "Electronics Enclosure" | "3.2573" | "0.925" |
| "Fairing + Polyutherane Foam" | "0.148" | "0.288" |
| "Lumen" | "0.05" | "1" |
| "Propeller" | "0.3603" | "1.88" |
| "Side Plate" | "0.5" | "0.97" |
| "Top Plate" | "0.1" | "0.97" |

Appendix B – Modelica Implementation of Field

The code for implementing the field is as follows:

```

Modelica.Mechanics.MultiBody.Forces.WorldF
orceAndTorque field(animation = false);
protected
  // Fields
  outer
UnderwaterRigidBodyLibrary.Fields.WaterFie
ld waterField;
  outer Modelica.Mechanics.MultiBody.World
world;
equation
  // equations of motion
  r_0 = frame_a.r_0;
  v_0 = der(r_0);
  a_0 = der(v_0);
  w_a =
Modelica.Mechanics.MultiBody.Frames.angula
rVelocity2(frame_a.R);
  // forces and torques due to fields
  b_f = waterField.waterBuoyantForce(d =
density, m = body.m);
  f_d = waterField.waterDragForce(v =
body.v_0 - Frames.resolve1(frame_a.R,
cross(r_CM, w_a)), mu = mu_d, A = A);
  t_d = Frames.resolve1(frame_a.R,
waterField.waterDragTorque(w = w_a, k =
k_d));
  // applying force and torques due to
fields
  field.force = b_f + f_d;
  field.torque =
cross(Frames.resolve1(frame_a.R, r_CM),
b_f) + t_d +
cross(Frames.resolve1(frame_a.R, r_CM),
f_d);
  connect(field.frame_b, body.frame_a);
  connect(frame_a, body.frame_a);

```

References

J. Evans, M. Nahon, Dynamics modeling and performance evaluation of an autonomous underwater vehicle, *Ocean Engineering*, Volume 31, Issues 14–15, 2004, Pages 1835–1858, ISSN 0029-8018, doi:10.1016/j.oceaneng.2004.02.006

McMillian, S., Orin, D. E., & McGhee, R. B. (1995). *DynaMechs: An object oriented software package for efficient dynamic simulation of underwater robotic vehicles.*

Otter, M., Elmquist H, Mattson S. E., “The New Modelica Multibody Library”, *Proceedings of the 3rd International Modelica Conference*, Linköping, 2003

Prestero, T. (2001). Development of a six-degree of freedom simulation model for the REMUS autonomous underwater vehicle. In *OCEANS, 2001. MTS/IEEE Conference and Exhibition* (Vol. 1, pp. 450-455). IEEE.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software* (Vol. 3, No. 3.2, p. 5).

da Silva, J. E., Terra, B., Martins, R., & de Sousa, J. B. (2007, August). Modeling and simulation of the lauv autonomous underwater vehicle. In *13th IEEE IFAC International Conference on Methods and Models in Automation and Robotics*. Szczecin, Poland Szczecin, Poland.

Tang, S. C. (1999). *Modeling and simulation of the autonomous underwater vehicle, Autolytus* (Doctoral dissertation, Massachusetts Institute of Technology).

Thiele, B., Beutlich, T., Waurich, V., Sjölund, M., & Bellmann, T. (2017, July). Towards a Standard-Conform, Platform-Generic and Feature-Rich Modelica Device Drivers Library. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017* (No. 132, pp. 713-723). Linköping University Electronic Press.

Tran M., Binns J., Chai S., Forrest A., Nguyen H. (2018) AUVSIPRO – A Simulation Program for Performance Prediction of Autonomous Underwater Vehicle with Different Propulsion System Configurations. In: Mazal J. (eds) *Modelling and Simulation for Autonomous Systems. MESAS 2017. Lecture Notes in Computer Science*, vol 10756. Springer, Cham

M. Triantafyllou. *2.154 Maneuvering and Control of Surface and Underwater Vehicles (13.49)*. Fall 2004. Massachusetts Institute of Technology: MIT OpenCourseWare, <https://ocw.mit.edu>. License: Creative Commons BY-NC-SA

Wadoo, S., & Kachroo, P. (2016). *Autonomous underwater vehicles: modeling, control design and simulation*. CRC Press.

Wang, C., Zhang, F., & Schaefer, D. (2015). Dynamic modeling of an autonomous underwater vehicle. *Journal of Marine Science and Technology*, 20(2), 199-212.

Wang, W., & Clark, C. M. (2006). Modeling and simulation of the VideoRay Pro III underwater vehicle. *Computer Science and Software Engineering*, 66.

Yang, R., Probst, I., Mansours, A., Li, M., & Clement, B. (2016). Underwater vehicle modeling and control application to ciscrea robot. In *Quantitative Monitoring of the Underwater Environment* (pp. 89-106). Springer, Cham.

Yuh, J. (1990). Modeling and control of underwater robotic vehicles. *IEEE Transactions on Systems, man, and Cybernetics*, 20(6), 1475-1483.

SystemModeler (2015) Copyright © 2015 Wolfram Research, Inc. <http://wolfram.com/system-modeler/>

BlueROV2, Blue Robotics, Inc. <http://docs.bluerobotics.com/brov2/>

Hybridisation and splitting of a crank angle resolved internal combustion engine model using a mean value intake for real-time performance

Xiaoran Han Alessandro Picarelli Mike Dempsey Romain Gillot

Claytex service limited, UK,

{xiaoran.han,alessandro.picarelli,mike.dempsey,romain.gillot}@claytex.com

Abstract

This paper describes splitting a crank angle resolved three cylinder combustion engine with an air path model and a combustion model. This is to distribute the computational effort on hardware by running models on separate cores to achieve real time capability. Hardware tests show the split models are not able to achieve real time because the thermal dynamics of air path model and combustion model are highly interconnected and computing the models on separate cores will introduce delay and solution can become inaccurate and even infeasible. In order to achieve real time capability while ensuring the results are accurate (2-5% percent max. error), a new method is proposed, in which instead of running with a complete fluid intake and exhaust model, the combustion model runs with a mean value intake model calibrated for many operating points across the speed-load range. The results show that the combustion model running with mean value intake model is able to produce highly accurate result and real time capability is achievable. By using mean value intake model, calibration effort is significantly reduced compared to purely table based method as the mean value model captures essential dynamics and is able to predict reliably between transition from one operating point to another. The mean value method takes into account Air Fuel Ratio (AFR) dynamics and thus calibration against AFR becomes unnecessary. Comparing to a non-mean value purely table based method, the latter requires calibration at densely scattered operating points in order for the transition between each calibration point to be smooth enough. In calibrating the mean value model a controller is designed to control the dynamics error to zero. This control based method shows high efficiency compared to optimization tools as it does not depend on initial values and iteration process of the calibrating parameters. A function is created to automatically create the tables calibrated. The calibrated mean value intake model is run with a combustion model on a Concurrent test/HiL rig and shows real time capability is achieved with good accuracy. The physical engine model is built in Dymola.

Keywords: mean value intake model, split engine model, automated calibration

1 Introduction

When running models to achieve real time in hardware-in-Loop applications, sometimes it is desirable to split a larger model into separate cores so that parallel processing features in the hardware can compute models simultaneously. However when models running on separate cores are highly interconnected, i.e. have feedback loops, where outputs of subsystem A are fed into subsystem B whose own outputs are in turn fed back as inputs to subsystem A, communication delays due to sampling within feedback loop can cause inaccuracy or infeasibility of the solutions, depending on the size of the delay or sampling frequency. In order to attain a reasonable accuracy of the solution, high frequency sampling is required to reduce the size this delay. However high frequency sampling will create more computational overheads potentially causing more overruns and hence render real time capability unattainable. To avoid feedback loops, a table based or neural network based method or other similar method can be used to replace the subsystems, where the inputs of the table or neural network are command signals at a higher level which does not require feedback from the subsystems. The tables or neural networks are calibrated off-line for each operating point using the inputs and outputs of the subsystems.

When there are lack of dynamical models of a system, the calibration points interval will need to be quite small in order to capture the nonlinear dynamics of the system between each operating point. The densely scattered operating points to be calibrated take a lot of effort to gather in the physical tests and are not always robust and accurate during transient from one operating point to another when simulated. It is therefore desirable to have dynamical models available where the essential dynamics of physical systems can be computed by the models rather than look-up tables. The dynamical models can be calibrated against physical system measurement where the identified parameters of the dynamical system are recorded in look-up tables. Thus the dynamics of the physical system during transition from one operating point to another is captured by the dynamical models. This increases the robustness and accuracy of the calibration as some of dynamical systems can be modeled so that they are robust against certain parameter variations. As a result of the dynamical system

being robust against parameter variation, this reduces the number of operating points to be calibrated.

For calibration, tools can be used which are gradient based and iterate a number of simulations before convergence criteria is fulfilled. The convergence of gradient based optimization depends on the initial values of the parameters to be calibrated and convergence of the criteria can not always be guaranteed after the end of simulation iterations. For calibration of multiple variables using multiple tuning parameters, where variables are interconnected, choosing initial values of the tuning parameters for convergence becomes even more challenging. This is because the convergence of one variable depends on the convergence of the other variables. Another advantage of having a dynamical model available is that controllers may be designed to ensure convergence of calibrated variables. The inputs of the controllers are the errors between the measured and calibrated variables and outputs of the controllers are tuning parameters. Controller gains can be designed such that errors starting with different initial values will be driven into the neighbourhood of zero. This method only requires one iteration to simulate in the presence of different initial values of calibrated variables, if controllers are designed properly.

Once calibration results are obtained it is time consuming to put them manually into look-up tables. This not only reduces efficiency but can create errors due to wrong data being put into the tables by the user. It can be also hard to debug which data has been wrongly entered to all the calibrated tables if there is an error. This would either require looking through all the tables and checking each of the entries against the calibration result, or creating test experiment to test each table with correct inputs to the table and check if outputs of the table are produced correctly. A more efficient way is to create a function that automatically puts the calibrated results into look-up tables without human intervention. This will increase efficiency and minimize potential user errors.

In this paper, a crank angle resolved three cylinder gasoline engine is considered for testing its real-time performance on a Concurrent real-time test rig. The engine model is required to be split into air path, i.e. intake and exhaust, and combustion using feedback loops so that they can be run on separate cores. It is shown that splitting the model with feedback loops is inefficient in achieving real time performance. Real time performance is jeopardised due to the delay feedback loops and is therefore not achievable with this level of detail. An alternative approach must be considered to eliminate the computational burdens caused by the feedback loops. Motivated by this problem, a look-up table approach becomes a first option for avoiding feedback loops. A mean value intake model is used for generating correct pressure and mass flow rate in intake manifold. Look-up tables are calibrated to generate throttle discharge coefficient and the volumetric efficiency of intake ports that are formulated in mean value intake model. Controllers are designed for the mean value

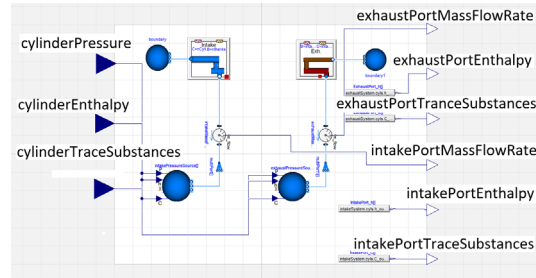


Figure 1. Split air path.

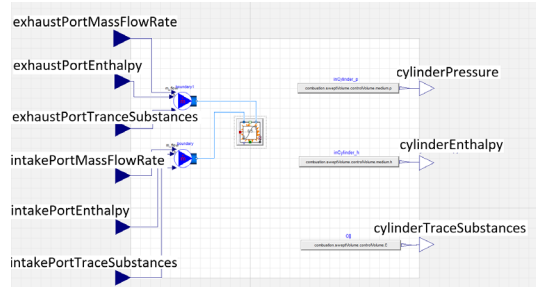


Figure 2. Split combustion.

intake model, whose inputs are mass flow rate and plenum pressure errors between the measurement and values calculated by the mean value model and outputs are the discharge coefficient and volumetric efficiency. It is shown that the mean value intake model calibrated with fewer operating points achieves similar accuracy as those calibrated with more operating points. In addition, the mean value based approach is shown to have much smoother transition phase over using purely table based approach which does not use dynamical models. A function is created to populate the entries of the tables with calibrated results automatically. Because the mean value model takes the effect of air fuel ratio (AFR) into account, calibration can be performed by choosing stoichiometric AFR for different throttle angle, engine speed, intake and exhaust phasing. This AFR dependence reduces calibration for different AFRs since its effect is considered in the model developed. An evaluation of the method, (mean value intake model with look-up tables as a replacement of a physical fluid based intake model with feedback loops to and from the combustion model) is carried out on a Concurrent real-time test rig to show that real-time performance is easily achievable

2 Splitting engine model with intake and combustion models

This section shows the splitting of an engine model into an air path model, i.e. intake and exhaust, and a combustion model. Figure 1 and 2 show splitted air path and combustion. Exhaust and intake model output *exhaustPortMassFlowRate*, *exhaustPortEnthalpy*, *exhaustPortTraceSubstances*, *intakePortMassFlowRate*, *intakePortEnthalpy*, *intakePortTraceSubstances*, which are the inputs of combustion model. The outputs of the

combustion model *cylinderPressure*, *cylinderEnthalpy*, *cylinderTraceSubstances* are the inputs of the air path model. The splitting method is described in detail in (Han, 2017a). The split models are tested on a Concurrent test rig for real-time performance evaluation by running the two models on separate cores. The test results reveal that splitting the models with feedback loops and running the models on separate core will slow down the simulation speed significantly and real time performance is not achievable. An excessively large sampling rate has to be chosen which can cause models to no longer respond correctly. In the following section, a mean value engine intake manifold model is introduced which calculate plenum pressure and mass flow rate at the intake port. The model is calibrated and verified so that it replaces the intake model in Figure 1 and calculates intake plenum pressure reliably.

3 Mean value engine intake model and its calibration using control design

Mean value engine models have been developed for modelling and control design (Heywood, 1988), (Guzzella and Onder, 2004). An idle control design of a crank angle resolved engine model using sliding mode and mean value engine model is described in (Han, 2017b) and (Han, 2017c). In this application, the model is calibrated at each operating point so that it can be used to replace the intake manifold as shown in figure 1. This section shows how to tune calibration parameters in the mean engine value model using control design.

3.1 Mean value model of intake mass flow rate and its calibration

Mass flow through the throttle for naturally aspirated engine can be approximated as (Guzzella and Onder, 2004)

$$\dot{m}_a(t) = \begin{cases} A_a \lambda_{CD} \frac{p_a}{\sqrt{RT_a}} \frac{1}{\sqrt{2}}, & \text{if } \frac{p_m(t)}{p_a} < 0.5, \\ A_a \lambda_{CD} \frac{p_a}{\sqrt{RT_a}} \sqrt{2 \frac{p_m(t)}{p_a} \left[1 - \frac{p_m(t)}{p_a}\right]}, & \text{else.} \end{cases} \quad (1)$$

where A_a is the fixed full area of the throttle, λ_{CD} is throttle discharge coefficient, which needs to be calibrated, p_a is the pressure upstream, ambient pressure, R is the ideal gas constant, T_a is the temperature upstream, p_m is manifold pressure, which is calculated in (5) and \dot{m}_a is the air mass flow through the throttle. λ_{CD} is tuned to calibrate (1) at each operating speed and load condition against a measured quantity. Calibration tools can be used to adjust λ_{CD} at each iteration based on an optimization criteria so that the value of the calculated mass flow rate \dot{m}_a is close to the measured mass flow rate, defined as $\dot{m}_{aMeasured}$. The number of iterations required before the optimization criteria is fulfilled depends on the convergence rate and the initial value of calibration parameters. A more efficient way to tune λ_{CD} by using the control method is presented. The method proves to be very efficient as dependence of the

convergence on the initial values of parameters is avoided by designing a controller which yields the feedback loop system to be a linear one. Simulation is only required to run once to determine the correct value of the calibrated parameters.

To design the controller, we define the error between measured air mass and calculated air mass through the throttle as

$$e_{air}(t) = m_{aMeasured}(t) - m_a(t) \quad (2)$$

where e_{air} is the error. A controller can be designed such that

$$\lambda_{CD} = \begin{cases} \frac{\sqrt{2RT_a}}{A_a p_a} K_m e_{air}(t), & \text{if } \frac{p_m(t)}{p_a} < 0.5, \\ \frac{\sqrt{RT_a}}{A_a p_a} \frac{K_m e_{air}(t)}{\sqrt{2 \frac{p_m(t)}{p_a} \left(1 - \frac{p_m(t)}{p_a}\right)}}, & \text{else.} \end{cases} \quad (3)$$

where K_m is a positive control gain. Note that there will always be a pressure drop from ambient pressure p_a to p_m in order for air flowing through throttle and then down to cylinder via intake valves. Substitute (3) into (1), equation (1) becomes

$$\dot{m}_a(t) = K_m (m_{aMeasured}(t) - m_a(t)) \quad (4)$$

It is easy to see that by choosing an appropriate control gain K_m , mass flowed through throttle can be controlled such that $e_{air} \rightarrow 0$ eventually and $\dot{m}_a \rightarrow \dot{m}_{aMeasured}$.

3.2 Mean value model of intake manifold pressure and its calibration

Manifold pressure can be modeled as

$$\dot{p}_m(t) = \frac{RT_m}{V_m} [\dot{m}_a(t) - \dot{m}_\beta(t)] \quad (5)$$

where T_m is the manifold temperature, V_m is the volume of intake manifold, \dot{m}_a is defined in (1) and \dot{m}_β is the mass flow rate into cylinder. Mass flow rate into cylinder can be modeled as

$$\dot{m}_\beta(t) = \frac{\dot{m}_e(t)}{1 + \frac{1}{\lambda(t)\sigma_0}}, \quad (6)$$

$$\dot{m}_e(t) = \frac{p_m(t)}{RT_m(t)} \lambda_l(p_m(t), \omega_e(t)) V_d \frac{\omega_e(t)}{4\pi}$$

where $\lambda_l(\cdot)$ is the volumetric efficiency of intake ports and valves, denoted as

$$\lambda_l(\omega_e(t), p_m(t)) = \frac{m_\beta(t)}{\rho_m(t)V_d} \quad (7)$$

where ρ_m is air density in intake manifold, V_d is the engine displacement volume, ω_e is engine speed, λ is the air fuel ratio

$$\lambda(t) = \frac{1}{\sigma_0} \frac{\dot{m}_\beta(t)}{\dot{m}_\phi(t)} \quad (8)$$

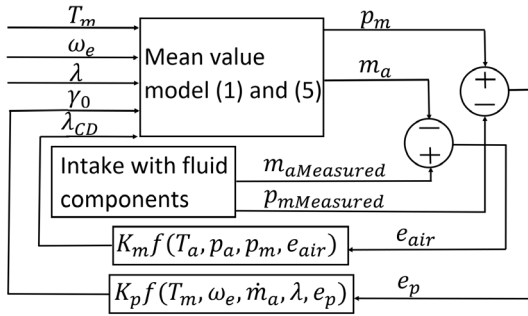


Figure 3. Schematic for control calibration of mean value intake model

where σ_0 is approximately 14.67 and \dot{m}_ϕ is fuel flow rate. Approximation of the volumetric efficiency of intake ports and valves can be formulated as

$$\lambda_l(p_m(t), \omega_e(t)) = \lambda_{lp}(p_m(t))\lambda_{lw}(\omega_e(t))$$

where

$$\lambda_{lp}(p_m(t)) = \frac{V_c + V_d}{V_d} - \left(\frac{p_{out}(t)}{p_m(t)}\right)^{\frac{1}{\kappa}} \frac{V_c}{V_d}$$

$$\lambda_{lw}(\omega_e(t)) = \gamma_0(t) + \gamma_1 \omega_e(t) + \gamma_2 \omega_e(t)^2 \quad (9)$$

where p_{out} is the exhaust manifold pressure, V_c is clearance volume, and κ , γ_0 , γ_1 and γ_2 are tuning parameters. For simplicity, we have chosen γ_0 as calibration parameter and γ_1 and γ_2 are fixed to be small values.

To design a controller for calibrating manifold pressure, the pressure error is defined as

$$e_p(t) = p_m(t) - p_{mMeasured}(t) \quad (10)$$

A controller can be designed such that

$$\gamma_0(t) = \frac{4\pi(1 + \frac{1}{\lambda\sigma_0})}{p_m(t)\lambda_{lp}V_d\omega_e(t)} (K_p V_m e_p(t) + RT_m \dot{m}_a(t)) - \gamma_1 \omega_e(t) - \gamma_2 \omega_e^2(t) \quad (11)$$

where K_p is a positive control gain. Substitute equation (11) into (9) and equation (9) into (6), equation (5) becomes

$$\dot{p}_m(t) = K_p (p_{mMeasured}(t) - p_m(t)) \quad (12)$$

By choosing an appropriate control gain K_p , $p_m(t) \rightarrow p_{mMeasured}(t)$ in (12) and $e_p(t) \rightarrow 0$. Figure 3 shows a schematic of on using controller design for calibrating mean value intake model, as described in this section.

3.3 Simulation results

This section shows calibration performance using controller design described above for 10% throttle opening, 1000rpm, stoichiometric AFR and default intake and exhaust phasing. Figure 4 shows the measured and calculated mean average and actual value of intake pressure and intake mass flow rate. Figure 5 shows control gains γ_0 and

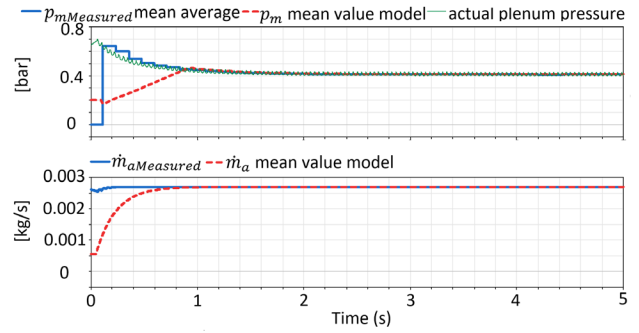


Figure 4. Calibration performance for calculated plenum pressure p_m and air flow rate through throttle \dot{m}_a

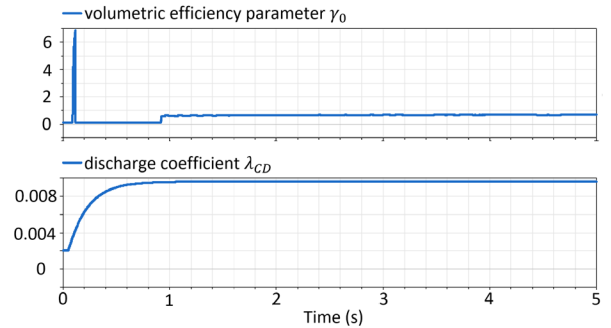


Figure 5. Calibration control gains γ_0 and λ_{CD}

λ_{CD} which are given in (11) and (3). Both calculated intake pressure p_m and calculated intake mass flow rate \dot{m}_a reach to their corresponding measurement $p_{mMeasured}$ and $\dot{m}_{aMeasured}$ within 2 seconds. Control gains $K_p = 5$ and $K_m = 20$ are chosen for computing γ_0 and λ_{CD} . Thus the proposed method for calibrating mean value air path engine model using control design is shown to be efficient and accurate. Controllers are able to control the calculated values despite their initial values being different from their measured values. The control gains K_m and K_p can be fixed for all calibrating points because the dynamics of feedback loops for mass flow rate and manifold pressure are only a function of these two gains (4) and (12). Initial values of $m_a(t)$ and $p_m(t)$ will not have an effect on convergence because the closed loop systems are linear and asymptotic stability of (4) and (12) are guaranteed. This is important in ensuring that calibration process only requires one time simulation for each calibration point and calibration performance can be guaranteed. Due to this feature, calibration process can be automated by running each calibration point sequentially and recording calibration parameters into the entries of look-up tables. It will be difficult to automate this process if optimization methods are used which iterate simulation a number of times until target criterias are met which is not always possible and different initial values of calibration parameters have to be tested, in this case initial values of λ_{CD} and γ_0 .

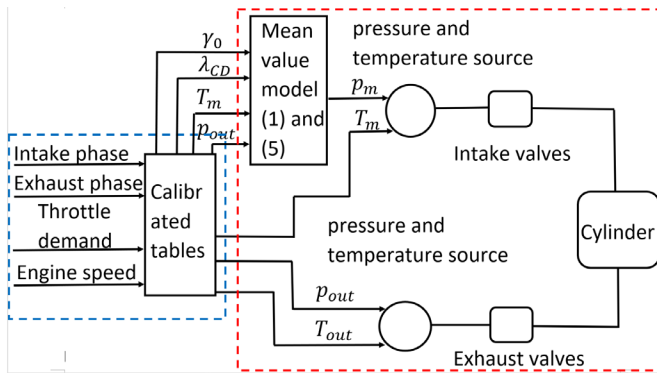


Figure 6. Calibrated tables and mean value model provide pressure temperature source for intake and exhaust valves

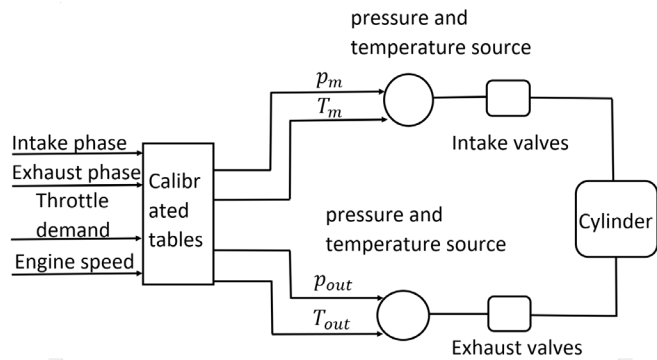


Figure 7. Calibrated tables provide pressure temperature source for intake and exhaust valves

4 Automation of calibration process

Automation of the calibration process described in section 3 can be performed by recording all the calibrating parameters into the entries of look-up tables. The model needs to be calibrated against each throttle demand, intake phasing, exhaust phasing, engine speed. Because the mean value model takes into account the AFR values in (6) and (9), only stoichiometric AFR is calibrated. Variation of AFR around stoichiometric will be compensated by the mean value model. The outputs of calibration are throttle discharge coefficient λ_{CD} in (3), volumetric efficiency parameters γ_0 in (11), intake manifold temperature T_m in (5), exhaust manifold pressure p_{out} in (9). These four parameters are needed to model mass flow rate \dot{m}_a in (1) and intake manifold pressure p_m in (5). The manifold pressure p_m , which is generated by the mean value model, together with calibrated manifold temperature T_m can be used to replace the intake manifold in Figure 1. In addition to the four outputs, exhaust manifold temperature, denoted as T_{out} also needs to be an output so that exhaust manifold in Figure 1 can be replaced by a pressure p_{out} and temperature T_{out} source, see Figure 6. Figure 7 shows calibration without using mean value model. The advantage of using mean value model based calibration over using non-mean value model based calibration will be shown in section 5.1.

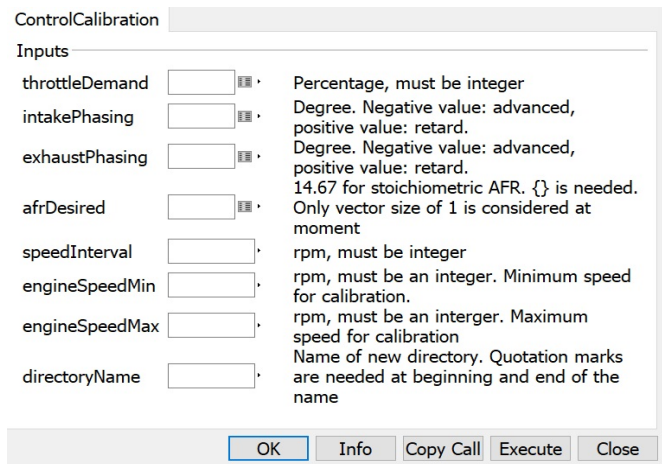


Figure 8. Function for automatic calibration where calibration points can be defined

A function can be created where users can define all the calibration points, Figure 8. For throttle demand, intake and exhaust phase shift, users will need to enter a vector of entries to be calibrated. Because the mean value model in (6) takes into account AFR variations, calibration against a stoichiometric AFR will be sufficient. This is one advantage of using mean value model as it reduces calibration dimensions by one. For engine speed, users will only need to enter minimum and maximum speed and speed interval step between the minimum and maximum speed. If residual of maximum speed over speed interval is not zero, only the last speed before the speed that exceeds the maximum speed by the residual is calibrated. The calibrated result is stored in a directory. Figure 9 shows automatically generated data records, tables, inputs and outputs of the calibrated tables, as shown in blue box in Figure 6. The number of columns of the tables is always five as it corresponds to the output number of the calibrated table. The number of rows of the tables depends on the engine speed points calibrated. In this case five rows of tables correspond to five engine speed points. The tables are of three dimensions and takes intake phase, exhaust phase and throttle as its three dimensional indexes. The values of calibrated parameters are stored in data records. In such way calibration of four dimensions by use of three dimensional tables can be implemented.

5 Verification of mean value intake model using simulation result

This section demonstrates the effectiveness of using a mean value intake model to predict intake manifold pressure. Automatic calibration as described in section 4 is performed to generate required tables. Simulations are carried out for different operating conditions and results are compared between engine models with a fluid component based intake manifold model and a mean value based intake manifold model. To show the advantage of using a mean value model for predicting the manifold pres-

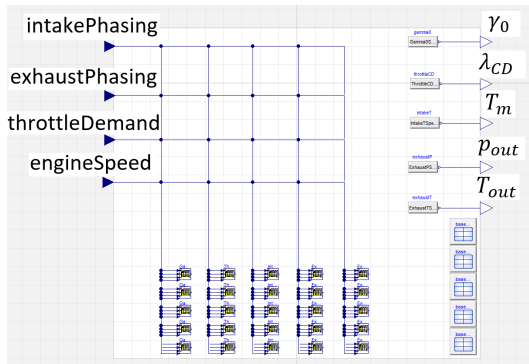


Figure 9. Automatically created calibration tables, data records and calibrating parameters

sure over using tables for predicting the manifold pressure, simulations are performed to show the difference in results. It has been mentioned in section 3 equation (6) and (7) that the mean value model considers the effect of AFR, so that calibration against different AFR is not needed. This feature is demonstrated by using the simulation results. Only one cylinder is used to produce the cylinder pressure. The cylinder pressure is sampled and delayed by the appropriate firing offset to represent the cylinder pressures from the other two cylinders. The cylinder pressures from all three cylinders are then applied to the pistons.

5.1 Comparison between the fluid based component intake, the mean value based intake model and the table based intake model between 5 to 20 percent throttle demand and 800 to 1000 rpm engine speed

The accuracy of the mean value model in predicting plenum pressures is compared with the fluid based component intake model. The advantage of using the mean value based intake model Figure 6 over using a table based intake model Figure 7 is shown in this section. Both mean value model based and table based calibration are performed at the same set of operating points according to Table 1. The actuation inputs for all three experiments, i.e. intake with fluid components, mean value model, tables without mean value model, are shown in Figure 10 to examine the transient performance. Fuel mass injected is fixed for all three experiments so that they are compared under the same fuel amount injected but making use of different intake components.

In Figure 11, the mean plenum pressure of engine models with fluid based component intake is compared with the mean value model based intake and non-mean value model with table based intake. It is shown that the manifold pressure from mean value mode based intake (red dotted) is produced smoothly during transients and matches very well with the pressure from fluid component based intake model (thick blue line). The plenum pressure of the non-mean value model table based intake (thin green line) however behaves more linearly and switches at inter-

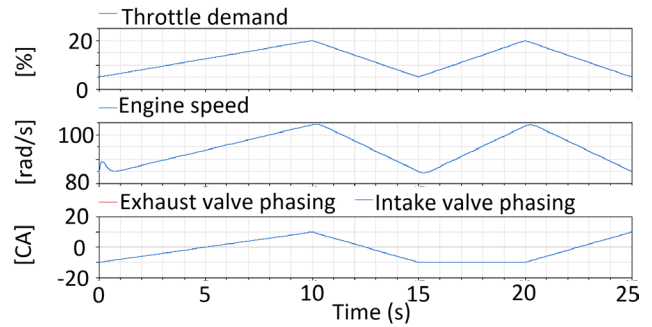


Figure 10. Inputs to calibration tables which are calibrated according to Table 1 with throttle demand 5 to 20 percent and engine speed 800 to 1000 rpm

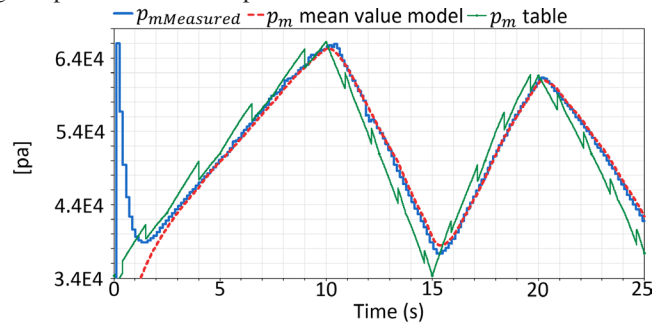


Figure 11. Plenum pressure from engine with fluid component based intake, mean value model based intake and table based intake

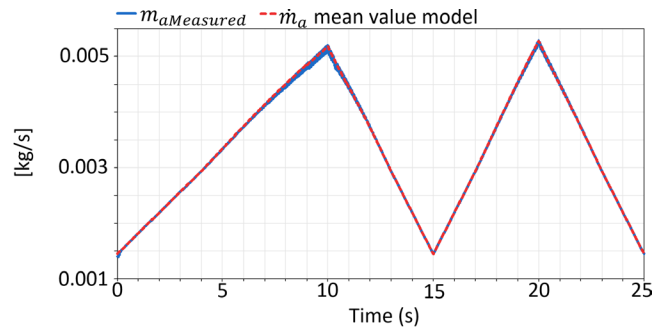


Figure 12. Air flow rate into cylinder with fluid component based intake and mean value model based intake

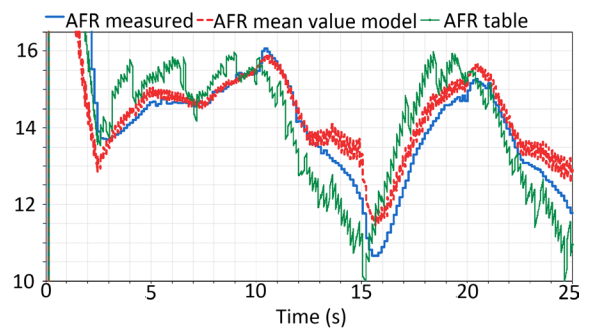


Figure 13. AFR from engine with fluid component based intake, mean value model based intake and table based intake

Table 1. Calibration points, throttle: 5 to 20 percent; engine speed: 800 to 1000 rpm

| Calibration inputs | calibration points |
|----------------------------|--------------------------|
| Throttle opening (percent) | 5, 8, 11, 14, 17, 20 |
| Engine speed (rpm) | 800, 850, 900, 950, 1000 |
| Intake phasing (CA) | -10, -5, 0, 5, 10 |
| Exhaust phasing (CA) | -10, -5, 0, 5, 10 |

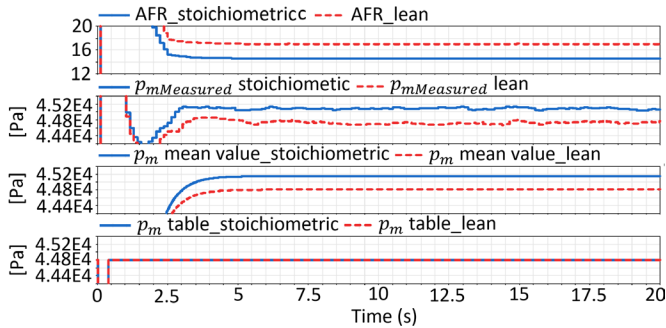


Figure 14. AFR effect on plenum pressure. Subplot 1: AFR values, subplot 2: mean average of plenum pressure from fluid component based intake, subplot 3: plenum pressure from mean value intake, subplot 4: plenum pressure from table based intake

sections between two different calibrated speed profiles. The accuracy is not as good at the result from mean value model. Intake mass flow rates from the fluid based component intake manifold and mean value model based intake manifold are shown in Figure 12. The mass flow rate is calculated by mean value model correctly. Figure 13 shows AFR that corresponds to plenum pressure in Figure 11 and air flow rate in Figure 12. It is shown that the mean value model based intake engine model tracks AFR which follows closely with AFR produced by fluid based component intake engine model, except after 10 second where the AFR from mean value model based intake model is overestimated compared to the AFR produced by the fluid component based intake model. The overestimated AFR is caused by overestimated plenum pressure in the mean value intake model, see Figure 11. To improve calibration precision, more calibration points can be chosen in this region. It is noted that improvement on the accuracy of the calibrated intake manifold temperature will improve the mean value model accuracy during transients. For different AFR values, between a minimum of 11 and a maximum of 16 in Figure 13, the mean value based intake model predict plenum pressures correctly in the presence of AFR variations.

The effect of AFR variation on the mean value model can be further illustrated in Figure 14. The engine is running at 10 percent throttle demand at 900 rpm. Intake and exhaust phasing are kept at default. For AFR values from stoichiometric to lean, blue solid to red dashed line in subplot 1 in Figure 14, mean average of plenum pres-

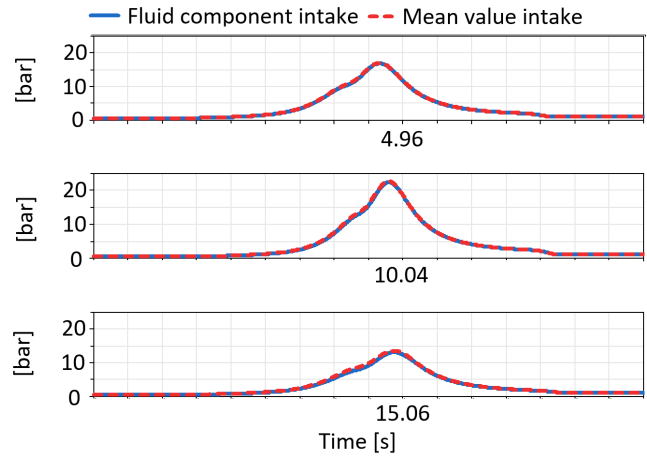


Figure 15. Cylinder pressure between fluid component based intake and mean value model based intake model at 4.96, 10.04 and 15.06 seconds, under plenum pressure in Figure 11

sure in the fluid based component intake model becomes lower, subplot 2 in Figure 14. This trend is reproduced by the mean value model of intake manifold, subplot 3 of Figure 14. The table based non-mean value intake model however produces the same manifold pressure regardless of AFR variations, subplot 4 of Figure 14.

For the fluid based component intake and mean value model based intake, plenum pressures produced by the two models are shown in Figure 11. Their corresponding cylinder pressures at 5s, 10s, 15s, 20s and 25s are selected and shown in Figure 15 and Figure 16. It is seen that with a mean value based intake, cylinder pressure at 5s and 10s matches very closely to cylinder pressure produced using fluid based component intake. At 15s, 20s and 25s cylinder pressure under mean value based intake is slightly higher than cylinder pressure under fluid component based intake. This is because under the same fuel mass injected the AFR for the mean value based intake is similar at 5s and 10s but slightly lean at 15s, 20s and 25s due to more air present in the cylinder, see Figure 13.

5.2 Comparison between fluid component based intake, mean value based intake model under other throttle demand and engine speed profiles

This section examines performance of the mean value model based intake under conditions other than calibration points in Table 1.

5.2.1 Throttle demand from 5 to 70 percent at engine speed from 800 to 1000 rpm

Table 2 shows calibrating points at 5 to 70 percent throttle demand and 800 to 1000 rpm engine speed. The calibrating point interval for throttle demand is 10 percent in Table 2, while it is 3 in Table 1. Calibration interval for intake and exhaust phasing is 10 CA in Table 2, while it is 5 CA in Table 1. This is to examine if the mean value

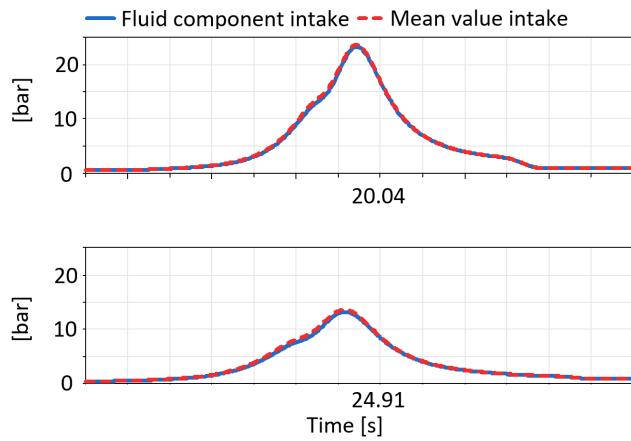


Figure 16. Cylinder pressure between fluid component based intake and mean value model based intake model at 20.04 and 24.91 seconds, under plenum pressure in Figure 11

model is still able to maintain good accuracy despite larger calibrating point intervals.

The actuation inputs for the experiments are shown in Figure 17, which shows different input patterns compared to Figure 10. Plenum pressure and AFR between fluid component based intake and mean value based intake are shown in Figure 18. It is seen that plenum pressure from mean value intake matches very closely to the mean of the plenum pressure produced by fluid based component intake, despite the fact that the mean value model is calibrated with larger calibration intervals. AFR values are similar between 5 and 15 seconds, where AFR changes between rich and lean. After 15 seconds, the AFR from the mean value intake is slightly lower than for the fluid based component intake.

Table 2. Calibration points, throttle: 5 to 70 percent; engine speed: 800 to 1000 rpm

| Calibration inputs | calibration points |
|----------------------------|-------------------------------|
| Throttle opening (percent) | 5, 10, 20, 30, 40, 50, 60, 70 |
| Engine speed (rpm) | 800, 850, 900, 950, 1000 |
| Intake phasing (CA) | -10, 0, 10 |
| Exhaust phasing (CA) | -10, 0, 10 |

5.2.2 Throttle demand from 5 to 60 percent at engine speed from 4100 to 5000 rpm

Table 3 shows calibration points for the throttle and speed range. Not that for throttle demand, calibration point increases by 3 percent after 10 but between 5 to 10 percent the increment is 5. Figure 19 and 20 show inputs and plenum pressure results. It is noted that the plenum pressure produced by the mean value engine matches very well with plenum pressure produced by fluid based component intake, except at very low load points between 8 and 12 seconds where plenum pressure predicted by the

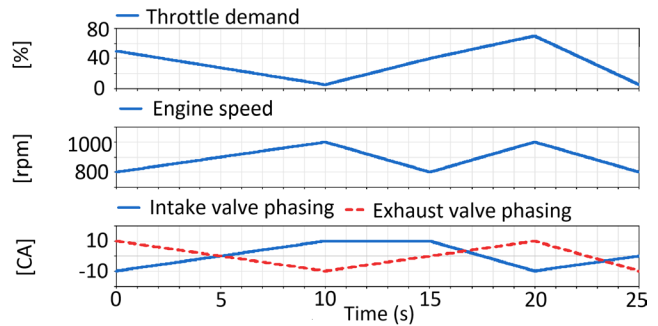


Figure 17. Inputs to calibration tables which are calibrated according to Table 2 with throttle demand 5 to 70 percent and engine speed 800 to 1000 rpm

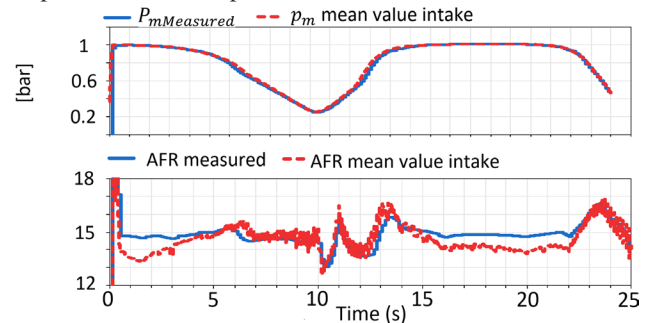


Figure 18. Mean average of plenum pressure and AFR from engine with fluid component based intake and mean value model based intake

mean value model is slightly higher than that produced by the fluid based intake model. This is because the calibration point between 5 and 10 percent throttle demand is not calibrated. This reveals that more throttle demand points should be calibrated for at very low load.

Table 3. Calibration points, throttle: 5 to 60 percent; engine speed: 4100 to 5000 rpm

| Calibration inputs | calibration points |
|----------------------------|-----------------------------|
| Throttle opening (percent) | 5, 10, +3, ..., +3, 61 |
| Engine speed (rpm) | 4100, +100, ..., +100, 5000 |
| Intake phasing (CA) | -10, -6, 0, 6, 10 |
| Exhaust phasing (CA) | -10, -6, 0, 6, 10 |

6 Hardware testing

6.1 Throttle demand 5 to 20 percent and engine speed 800 to 1000 rpm

A quad-core Concurrent test rig, each core having 2.5GHz clock rate, with 3.9 GB Ram and RedHawk Linux operating system is used for evaluation of the real time performance of the models. Mechanical components (such as the crankshaft) and controller which generates injection, ignition, speed, throttle, intake and exhaust phasing are in one model. The calibrated tables are also within the

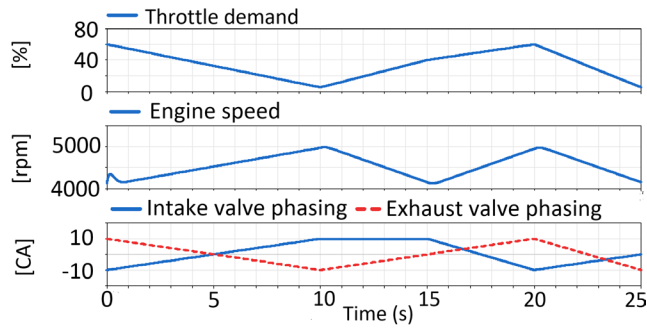


Figure 19. Inputs to calibration tables which are calibrated according to Table 3 with throttle demand 5 to 60 percent and engine speed 4100 to 5000 rpm

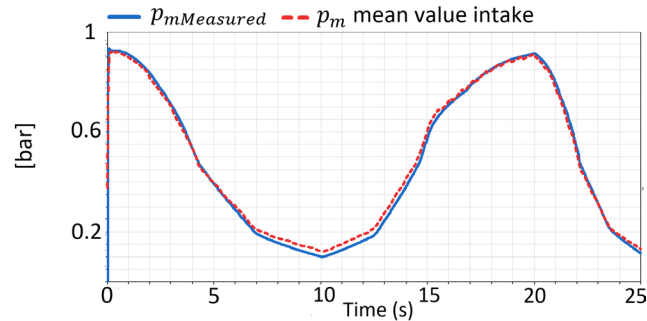


Figure 20. Mean average of plenum pressure from engine with fluid component based intake and mean value model based intake

same model, see blue dashed box in Figure 6. Mean value model, intake and exhaust pressure source, intake and exhaust valves and cylinder are put into another model, see red dashed box in Figure 6. The experiment run on the hardware is the same experiment shown in Figure 10 to Figure 13, where cylinder pressures at different time instants are shown in Figure 15 and Figure 16. Figure 21 to Figure 25 show cylinder pressure recorded on hardware at the same set of time instants studied in Figure 15 and Figure 16. It can be seen that the cylinder pressures produced on the Concurrent rig are of correct values. The execution frames for two models on separate cores are shown in Figure 26, where execution frame equals to 100 micro seconds. It has been tested that the models run in real time with execution frames equal to 500 micro seconds while keeping good accuracy.

6.2 Throttle demand 5 to 80 percent and engine speed 3500 to 6000 rpm

Real time performance can be achieved for the engine running at speed between 3500 rpm to 6000 rpm. Figure 27 and Figure 28 show simulation results in Dymola and Concurrent for throttle demand at 65 percent and an engine speed of 5300 rpm by setting the execution frames to be 150 micro seconds. There is a slight fluctuation in the cylinder pressure produced by Concurrent at real time but the accuracy is within a good margin.

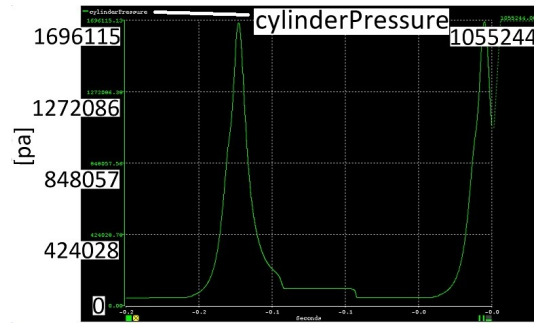


Figure 21. Cylinder pressure at 5s

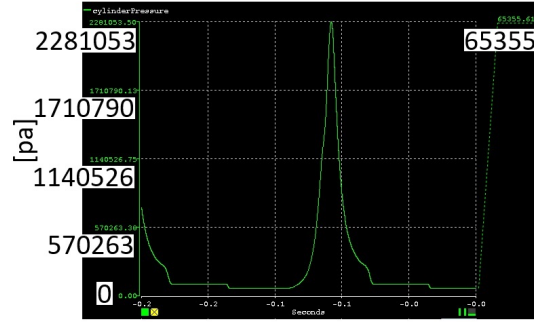


Figure 22. Cylinder pressure at 10s

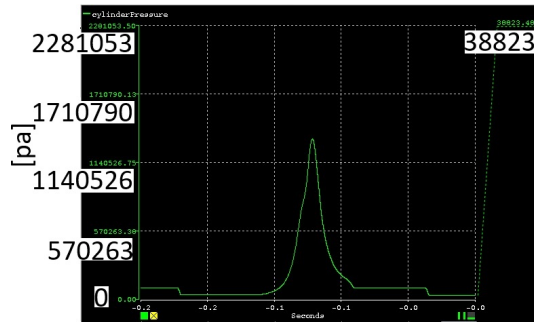


Figure 23. Cylinder pressure at 15s

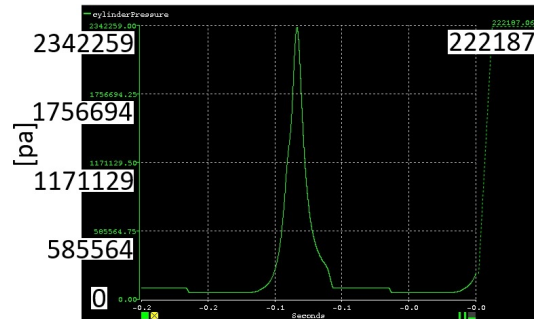


Figure 24. Cylinder pressure at 20s

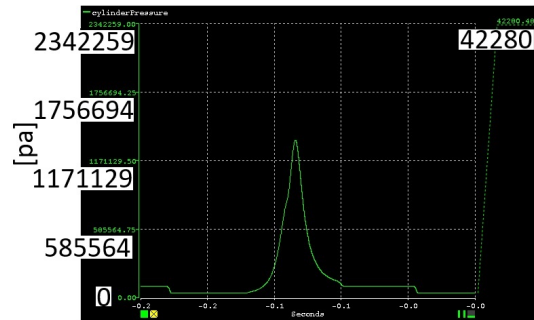


Figure 25. Cylinder pressure at 25s

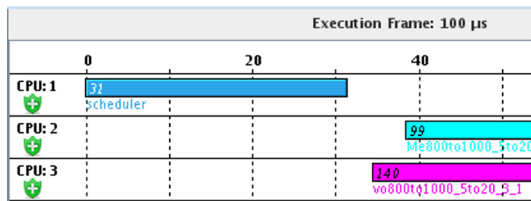


Figure 26. Execution frame for mechanical model on core 2 and cylinder volume model on core 3 at 5

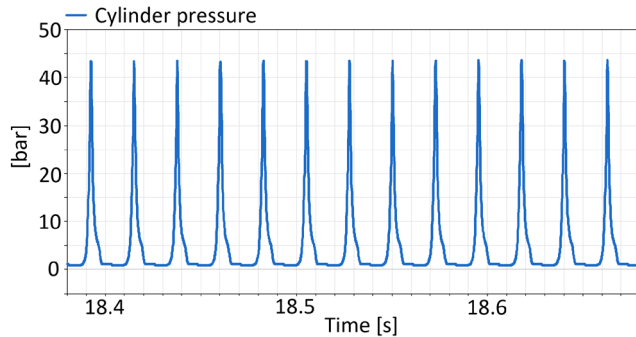


Figure 27. Cylinder pressure at 65 percent throttle demand and 5300 rpm produced in Dymola

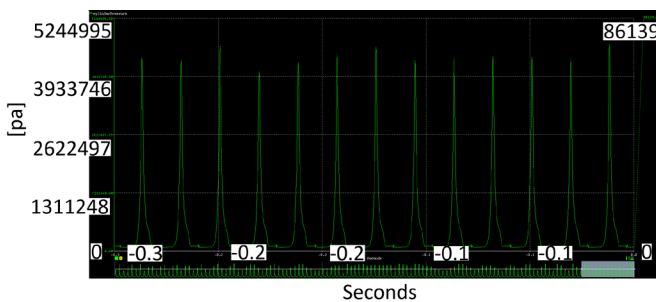


Figure 28. Cylinder pressure at 65 percent throttle demand and 5300 rpm produced in Concurrent

7 Conclusion

It has been shown that the mean value intake model approach is efficient to be used for crank angle resolved engine simulation in Hardware in Loop testing for real time performance. Automated calibration is efficient and accurate. By using a mean value model intake, calibration against AFR is not needed and the number of dimensional inputs for calibration becomes one less. The result produced by the mean value model is smoother than purely table based methods since plenum pressure is predicted by dynamical models rather than tables.

References

- L. Guzzella and C.H. Onder. *Introduction to modelling and control of internal combustion engine systems*. Springer-Verlag Berlin Heidelberg, 2004. ISBN 978-3-642-10775-7.
- X. Han. [Splitting mechanical, fluid ports and thermal ports using real inputs and real outputs](http://www.claytex.com/tech-blog/splitting-mechanical-and-fluid-devices-using-real-inputs-real-outputs/). <http://www.claytex.com/tech-blog/splitting-mechanical-and-fluid-devices-using-real-inputs-real-outputs/>, 2017a.
- X. Han. [Linearised Mean Value Engine Model-based Idle Control](http://www.claytex.com/blog/linearised-mean-value-engine-model-based-idle-control/). <http://www.claytex.com/blog/linearised-mean-value-engine-model-based-idle-control/>, 2017b.
- X. Han. [Idle speed control using model based sliding mode control](http://www.claytex.com/blog/idle-speed-control-using-model-based-sliding-mode-control/). <http://www.claytex.com/blog/idle-speed-control-using-model-based-sliding-mode-control/>, 2017c.
- J.B. Heywood. *Internal combustion engine fundamentals*. McGraw-Hill, Inc, 1988.

Component-Based 3D Modeling of Dynamic Systems

Andrea Neumayr¹ Martin Otter¹

¹DLR, Institute of System Dynamics and Control, Germany, {andrea.neumayr,martin.otter}@dlr.de

Abstract

The objective is to model and simulate larger and more complex 3-dimensional systems as it is possible with a pure equation-based modeling system such as Modelica. The approach shall combine component-based 3D modeling, as used in modern game engines, with equation-based modeling. The proposed methodology has been evaluated and tested in the experimental modeling environment Modia3D that is implemented with the Julia programming language.

Keywords: Modelica, Modia, Modia3D, Julia, DAE, equation-based modeling, component-based modeling, multibody, collision handling

1 Introduction

The objective is to model and simulate larger and more complex 3-dimensional systems as it is practically possible with a pure equation-based modeling system such as the current Modelica language version 3.4. Issues are:

- The data structures of an equation-based modeling system are limited as compared to a programming language such as C++ or Julia. For example, it is virtually impossible to define 3D meshes and collision handling algorithms in Modelica.
- Specialized operations in the 3D world are hard to use, such as to remove redundant constraints of a planar loop automatically, solve kinematic loops analytically, or use an $O(n)$ multibody algorithm. In Modelica, a user has to explicitly model such situations with specialized elements or use a pre-processor that generates Modelica code, see e.g. (Elmqvist et al., 2009).
- Since Modelica compilers expand the models for the symbolic engine, the same equation is analyzed many times. Thus, the number of expanded equations grows at least linearly with the number of model instances and therefore the compilation time grows at least linearly with the model size.

The goal of this article is to propose an approach how to combine 3D modeling techniques with equation-based modeling à la Modelica. This procedure has been evaluated and tested with the open source prototype *Modia3D*¹ (version 0.2.0-beta.1). It is implemented with the Julia programming language² (Bezanson et al., 2017) taking

¹<https://github.com/ModiaSim/Modia3D.jl>

²<https://julia.org>

advantage of Julia's powerful language features such as multiple dispatch and set-based types³. *Modia*⁴ (Elmqvist et al., 2016, 2017) shall be used for the equation-based modeling. The intention is to utilize the results of this prototyping in the design of the next Modelica language generation.

Modia3D has no graphical user interface. It would be useful to have 3D schematics as proposed by (Elmqvist et al., 2015a). The textual representation of *Modia3D* is designed for 3D schematics and not for Modelica 2D schematics. *Modia3D* provides a generic interface to visualize simulation results with different 3D renderers. Currently, the free community edition as well as the professional edition of the *DLR Visualization* library⁵ (Bellmann, 2009; Hellerer et al., 2014) are supported.

2 Component-Based 3D Modeling

Modern game engines, such as Unity or Unreal Engine, have a *component-based* design, so the architecture is based on *composition* and *aggregation*. Basically, in this context a coordinate system is located in the 3D world that has a container of optional components (such an object is called *GameObject*⁶ in Unity, *Actor*⁷ in Unreal Engine, *Object3D*⁸ in Three.js). Each of these components has properties such as geometry, visualization, dynamics, collision properties, light, camera, sound, etc., see e.g. (Nystrom, 2014)⁹. This design has the advantage that many optional components and variants can be defined and treated in a very flexible and unified way. In this paper, this very special variant of the generic *component-based design pattern* is called *component-based 3D modeling*.

Modelica 3.4 supports component-based design via *replaceable* components. Unfortunately, this language construct has limitations and is not sufficient for component-based design as needed below. On the other hand, Julia is particularly designed to support this programming pattern¹⁰ and is thus very well suited for the implementation of *Modia3D*.

³<https://docs.julialang.org/en/stable/manual/types/>

⁴<https://github.com/ModiaSim/Modia.jl>

⁵<https://visualization.ltx.de/>, <http://www.systemcontrolinnovationlab.de/the-dlr-visualization-library/>

⁶<https://docs.unity3d.com/Manual/GameObjects.html>

⁷<https://docs.unrealengine.com/en-us/Engine/Components>

⁸<https://threejs.org/docs/index.html#api/core/Object3D>

⁹<http://gameprogrammingpatterns.com/component.html>

¹⁰<http://www.stochasticlifestyle.com/type-dispatch-design-post-object-oriented-programming-julia>

2.1 Object3D

In Modia3D, component-based 3D modeling is performed with `Object3D` objects. An `Object3D` object consists of a 3D coordinate system that has associated, optional properties collected in the data container (see Figure 1). The

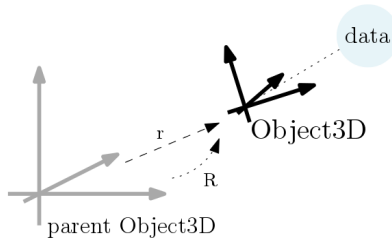


Figure 1. Object3D defined relatively to its parent.

code-snippet¹¹ of the following constructor call¹² creates a new `Object3D` object `obj`:

```
1 obj = Object3D(parent, data, r=[0,0,0],
2           R=[1 0 0;0 1 0;0 0 1],fixed=true)
```

Hereby, `obj` is defined relative to a `parent object3D`, with the position vector r and the rotation matrix R . It is rigidly connected to its `parent` if `fixed=true`, and can move freely if `fixed=false`. In the latter case, initial position and rotation matrix is defined with r, R .

Argument `data` is of the abstract type `AbstractObject3Ddata`. Therefore, all objects can be used which are a subtype of this type. There are further constructor functions for `Object3D`, therefore the arguments `parent` and `data` are also optional (e.g. line 3). An `Object3D` is said to be a reference `Object3D`, if no `parent Object3D` is given. The world-`object3D` can be defined as, for example

```
3 world = Object3D().
```

In Figure 2, the current abstract and concrete subtypes of `AbstractObject3Ddata` are shown. Instances of the concrete subtypes can be used for the positional argument `data`. In Figure 2, the concrete types are printed in light blue, abstract types in black, and types that are currently under implementation are printed in grey color. The most important concrete types are discussed below. Note, a `data` object consists of a set of optional components, providing in a flexible way variants and different functionality. All these components are positioned and moved with the same concept - the coordinate system to which the components are attached. The conceptual difference to current Modelica is that the `Modelica.Mechanics.MultiBody` library defines coordinate systems and properties (such as visualization data) with respect to various *Part* objects. As a result, the equations to define coordinate systems relative

¹¹For better reference every code-snippet is marked with a unique line number on the left-hand side.

¹²When calling a Julia function, all optional keyword arguments (name-value pairs) can be given in any order. They are set after the positional arguments (here: `parent` and `data`).

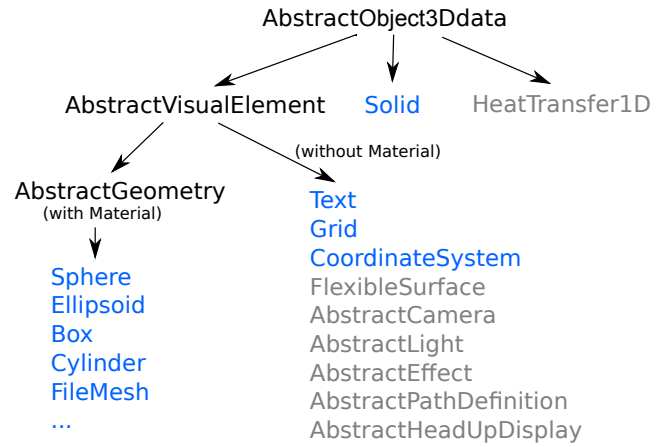


Figure 2. Overview of `AbstractObject3Ddata` types.

to each other are present many times in many different components, whereas in Modia3D these equations are present only once in the `Object3D` definition (and `Object3D` objects support much more flexible part definitions).

2.2 Visual Objects

Visualization objects are subtypes of `AbstractVisualElement`, which is also a subtype of `AbstractObject3Ddata`. These elements are used for animation purposes only. Basically, their Julia implementation is an interface to the *DLR Visualization* library (Bellmann, 2009; Hellerer et al., 2014). The concrete types which have a geometry and associated visualization properties are subtypes of `AbstractGeometry`. Their material is defined with a `Material` object. For example, the following constructor call generates a new `Material` object:

```
4 vmat = Material(color=[0,0,255],
5           wireframe=false,transparency=0.5,
6           shininess=0.7,reflectslights=true)
```

Concrete subtypes with a geometry and a material are shown in Figure 3.

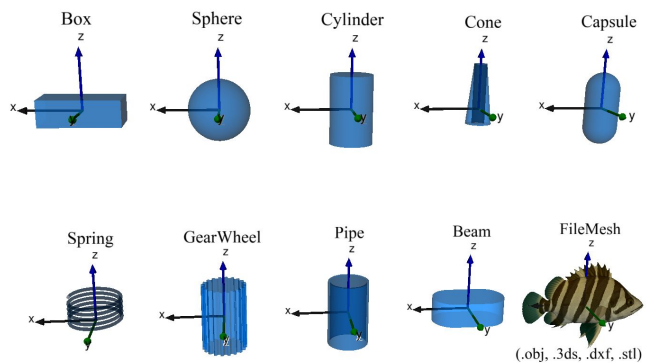


Figure 3. Visual elements with `Material`.

The following example defines an `Object3D`, which is positioned at $[0,0,0.8]$ in the world-`object3D` (line 3), is displayed with the visualization material `vmat` (lines 4 - 6), and has a sphere geometry with diameter = 0.9 m.

```

7 sphere = Object3D(world,
8           Sphere(0.9,material=vmat),
9           r=[0,0,0.8])

```

Additionally to the subtypes of `AbstractGeometry`, `Modia3D` supports currently the concrete types shown in Figure 4. These types do not have a `Material` object. Here, a grid with 0.7 m length and 0.6 m width, is defined.

```

10 grid = Object3D(world,Grid(0.7,0.6))

```

It is positioned at the origin of the world-object3D.

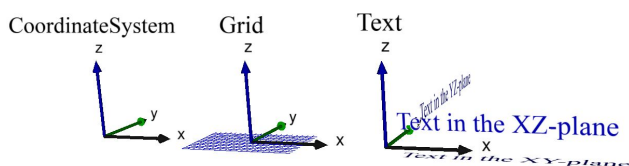


Figure 4. Visual elements without `Material`.

Remark 1. All objects which are subtypes of `AbstractVisualElement` are mutable objects. Therefore, they still can be changed after instantiation, especially during simulation.

2.3 Solid Objects

The type `Solid` is directly derived from `AbstractObject3Ddata` and defines solid physical objects. A solid object can have geometry, mass properties, can be visualized and can be used in collision handling, and all of these properties are optional. `Solid` objects are immutable to guarantee constant mass properties during simulation. The following constructor call creates a new `Solid` object.

```

11 solid = Solid(geo,massProperties,material,
12              contactMaterial=nothing)

```

The arguments have the following meaning:

`geo` defines the geometry of the solid. It is either `nothing`¹³ (= no geometry defined) or it is a subtype of `AbstractSolidGeometry`.

`massProperties` defines the mass properties of the solid. It is either `nothing` (= is massless) or there are various options to define these properties (see lines 19 - 21 below).

`material` defines the visualization properties of `geo`. It is either `nothing` (= `geo` is not visualized) or it is a `Material` object (e.g. lines 4 - 6).

`contactMaterial` defines the contact response characteristics of `geo`. It is either `nothing` (= `geo` is not included in the collision handling) or it is a subtype of `AbstractContactMaterial`.

Since all these properties are optional, there is a great flexibility to define the desired solid. Below, more details about the arguments of the solid constructor are given.

¹³In Julia, value `nothing` marks an empty value.

Argument: `geo`

Solid geometry objects `geo` are subtypes of the abstract type `AbstractSolidGeometry` and are shown in Figure 5. For example, a `SolidFileMesh` object can be defined with the following constructor call.

```

13 mesh = SolidFileMesh("pascal.obj",0.2)

```

Assuming that a file in `obj`-format is available as "pascal.obj" and the mesh shall be scaled by a factor of 0.2.

Currently, `Modia3D` supports collision handling only for *convex* objects. If a `SolidFileMesh` object is concave, collision handling is performed with respect to the convex hull of the mesh. Alternatively, the open source V-HCAD library¹⁴ can be used to approximate a concave mesh by a set of convex parts. Then, `Modia3D` utilizes these convex parts in collision handling and the original concave mesh for non-collision operations.

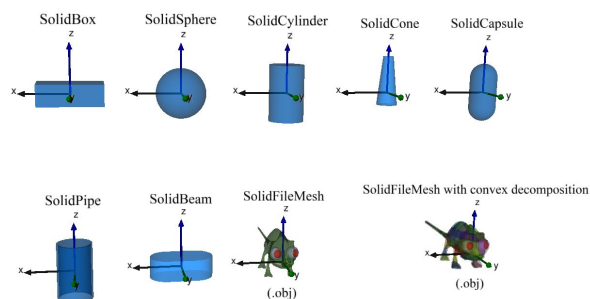


Figure 5. Solid geometry types.

The following functions¹⁵ compute key properties for rigid-body computations or collision handling and they are provided for all solid geometry objects displayed in Figure 5.

`volume(geo)` returns the volume of a solid geometry object `geo`.

`centroid(geo)` returns the position of the centroid of `geo`. If the solid is homogeneous, the centroid's position is identical to the center of mass.

`inertiaMatrix(geo,mass)` returns the inertia matrix of a solid geometry object `geo` with mass `mass`.

`boundingBox!(geo,...)` updates the bounding box of `geo`. This operation is used for collision handling (see below).

`supportPoint(geo,...)` returns the support point of `geo`. This operation is a key property for collision handling (see below).

In the following example some geometric properties of a `SolidSphere` object with diameter `D` are computed with the above mentioned functions.

¹⁴V-HCAD: <https://github.com/kmammou/v-hacd>

¹⁵As usual in Julia, function names with a ! at the end indicate that one or more of the input arguments are changed by the function call.

```

14 D      = 0.3; density = 7700
15 geo    = SolidSphere(D)
16 V      = volume(geo)
17 m      = density*V
18 IM     = inertiaMatrix(geo,m)

```

Currently, only mesh-data from wavefront (*.obj) files are supported. It is planned to generalize the support of meshes as proposed in (Elmqvist et al., 2015b), to directly define them in Modia3D and provide CSG (Constructive Solid Geometry) operations on them.

Argument: massProperties

There are several variants to define the optional mass properties: mass, center of mass, and inertia matrix. Examples of the different variants are:

```

19 mesh1 = Solid(SolidFileMesh(..), "Aluminium")
20 mesh2 = Solid(SolidFileMesh(..), 2.1)
21 massProp = MassProperties(m=2.1, Ixx=0.1, ..)
22 mesh3 = Solid(SolidFileMesh(..), massProp)

```

In the first case a string is given (line 19), such as "Aluminium". This string is a key in a dictionary in which some key data of materials is stored, such as density, Youngs modulus, heat capacity, and thermal conductivity. The density of the material is used together with the geometry `geo` to compute the needed mass properties (see lines 14 - 18). Alternatively, a number (line 20) can be given that is interpreted as the mass of the solid. Again, together with the geometry `geo` the needed mass properties are calculated. Finally, also an instance of type `MassProperties` (line 21) can be provided, in which all mass properties are explicitly given.

Argument: contactMaterial

The contact material `cmat` (line 23) defines how a solid behaves in contact cases. At the moment elastic contacts can be handled, with a spring - damper module. Therefore, a spring constant `c` and a damper constant `d` can be provided, as shown in the next example.

```

23 cmat    = ElasticResponse(c=1e5, d=100.0)
24 sphere  = Solid(SolidSphere(0.2), "Aluminium",
25               contactMaterial=cmat)

```

Example

A few examples are shown how solid objects can be defined:

```

26 geo     = SolidSphere(0.2)
27 vmat    = Material(color=[0,0,255],
28                  transparency=0.5)
29 cmat    = ElasticResponse(c=1e5, d=100.0)
30 basicSphere = Solid(geo, "Aluminium", vmat,
31                  contactMaterial=cmat)
32 sphere1 = Object3D(world, basicSphere,
33                  r=[1.0,0.0,0.0], fixed=false)
34 sphere2 = Object3D(world, basicSphere,
35                  r=[0.0,1.0,0.0], fixed=false)
36 sphere3 = Object3D(world, basicSphere,
37                  r=[0.0,0.0,1.0], fixed=false)

```

In the example above, a sphere `basicSphere` is defined that has a diameter of 0.2 m and is made of Aluminium. It is visualized with material `vmat`, and takes place in collision handling using contact material `cmat` for the response calculation. This definition is used to declare three spheres: `sphere1`, `sphere2`, `sphere3`. These spheres can move freely in space and are initially placed at different positions in the world-object3D (line 3). Note, although three spheres are declared, all the position-independent properties of the spheres, like visualization material, contact material etc. are defined only *once* by the reference object `basicSphere`. In Modelica, one could construct something similar by using *replaceable record constructors in modifiers*. The conceptual difference is that the *data* and *equations* of a `basicSphere` Modelica model would be present *three times* in the generated code and not *once* as in the Modia3D code.

If only one sphere shall be defined, the above definition (line 26 - 32) can also be given without auxiliary variables:

```

38 sphere = Object3D(world,
39   Solid(SolidSphere(0.2), "Aluminium",
40     Material(color=[0,0,255],
41             transparency=0.5),
42     contactMaterial=
43       ElasticResponse(c=1e5, d=100.0)),
44   r=[0.0,0.0,1.0], fixed=false)

```

2.4 Operations on Object3D

There are several functions that operate on `Object3D` objects, such as:

`isVisible(object3D, renderer)` returns true, if the data object associated with the `object3D` can be visualized (e.g. a solid-object where `geo` and `material` are defined) *and* the visualization element is supported by the utilized renderer

`hasMass(object3D)` returns true, if mass properties are associated with the `object3D`.

`canCollide(object3D)` returns true, if an `AbstractSolid` object is associated with the `object3D`, together with an `AbstractContactMaterial` object.

Depending on the underlying types of the elements of an `Object3D` object, type-specific methods are called (based on Julia's multiple dispatch feature).

3 Assembly Objects

In the previous section it was shown how to define `Object3D` objects and how to associate properties to an `Object3D` in a very flexible manner. In this section the aggregation of `Object3Ds` is discussed.

Hierarchical structures are defined with the Modia3D macro `@assembly` (lines 45 - 48). A Julia macro is a metaprogramming construct of Julia¹⁶. It generates an

¹⁶<https://docs.julialang.org/en/stable/manual/metaprogramming/>

abstract syntax tree of Julia code that is automatically compiled and executed at the line where the macro is called.

```
45 @assembly AssemblyName(..) begin
46   name = constructor(..)
47   < other statements >
48 end
```

The `@assembly` macro generates a new Julia type `AssemblyName` (it is a mutable struct) that (a) contains all left-hand side "name" definitions as elements, (b) uses the code of the `@assembly` for the constructor function for its struct, and (c) initializes support for hierarchical names of the elements of this new type. For example,

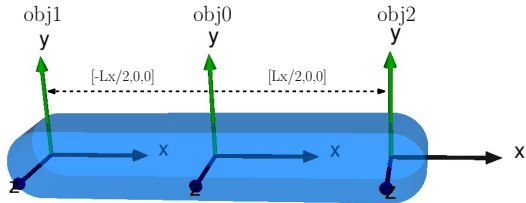


Figure 6. A solid bar with two additional Object3Ds.

the solid bar of Figure 6 consists of a beam element with two additional Object3Ds. This can be achieved with the following declarations:

```
49 @assembly Bar (;Lx=0.1, Ly=Lx/5, Lz=Lx) begin
50   obj0 = Object3D(Solid(SolidBeam(Lx, Ly, Lz),
51     "Aluminium",
52     Material(color="Blue")))
53   obj1 = Object3D(obj0, r=[-Lx/2, 0.0, 0.0])
54   obj2 = Object3D(obj0, r=[ Lx/2, 0.0, 0.0])
55 end
56 bar = Bar(Lx=1.0)
57 visualizeAssembly!(bar)
```

The reference Object3D `obj0` (line 50) is defined as a solid with a `SolidBeam` geometry. The two other Object3Ds - `obj1`, `obj2` (lines 53 - 54) - have `obj0` as parent Object3D and their positions are defined according to Figure 6. To check this definition, an instance of the new `Bar` type is constructed (line 56) and it is an input argument of the function call `visualizeAssembly!(bar)` that visualizes the assembly with the default renderer.

Since all left-hand side variables of `@assembly Bar` are elements of the new type, these elements can be accessed via the `bar` instance (line 56) as, e.g. `bar.obj1`. Since the code of an `@assembly` definition is used as a *constructor function*, order matters and thus the statements are executed in the given order¹⁷.

Assembly objects can also be elements in other assembly objects and therefore hierarchical structures can be built. For demonstration, the following planar *four-bar* mechanism¹⁸ is defined. It consists of three bars and the ground

¹⁷The main reason of this property is to have a simple implementation of the `@assembly` macro. With a more involved implementation, the definition between `begin ... end` could be given in any order and the constructor function could be generated from the sorted statements, provided no algebraic loops are present.

¹⁸https://en.wikipedia.org/wiki/Four-bar_linkage

(= the fourth bar) connected by four revolute joints forming a *planar* kinematic loop (see Figure 7).

```
58 @assembly Fourbar (;Lx=0.1) begin
59   world = Object3D(CoordinateSystem(0.6))
60   pos1 = Object3D(world, r=[Lx/2, 0.0, Lx/2])
61   pos2 = Object3D(pos1, r=[Lx, 0.0, 0.0])
62   ground = Object3D(world, Box(..), ..)
63   bar1 = Bar(Lx=Lx)
64   bar2 = Bar(Lx=Lx)
65   bar3 = Bar(Lx=Lx)
66   rev1 = Revolute(pos1, bar1.obj1,
67     phi_start=pi/2)
68   rev2 = Revolute(bar1.obj2, bar2.obj1,
69     phi_start=-pi/2)
70   rev3 = Revolute(bar3.obj2, bar2.obj2,
71     phi_start=-pi/2)
72   rev4 = Revolute(pos2, bar3.obj1,
73     phi_start=pi/2)
74   ...
75 end
76 fourbar = Fourbar(Lx=1.0)
77 visualizeAssembly!(fourbar)
```

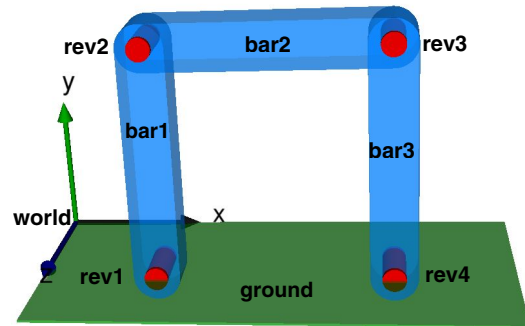


Figure 7. Planar four-bar mechanism.

A revolute joint is defined, with the constructor

```
78 Revolute(object1, object2) .
```

It constrains `object2` (line 78), so that the z-axis of `object2` coincides with the z-axis of `object1` (line 78) and can rotate around it. Via additional keyword arguments, the joint can be configured further. For example, `phi_start = angle` initially rotates `object1` along its z-axis for the given angle to arrive at `object2`. The revolute joints are visualized in Figure 7 with red cylinders.

Note, in Modelica and Modia a user has to treat one of the revolute joints differently. For example defining one of them to be a revolute cut-joint in a planar loop (Modelica model: `RevolutePlanarLoopConstraint`), since otherwise a redundant set of equations would be generated that cannot be handled with current symbolic engines.

Contrary, in Modia3D no special action is needed by the user. Instead, there is the requirement that the configuration defined by the assembly constructor must be *consistent*. For example, if `phi_start = pi` would be defined for `rev4`, then this start angle would not be consistent to the already defined configuration, and an error would occur. However, it would be possible to define the angle as `phi_start`

= NaN (= Not-a-Number), and the `Revolute` constructor of `rev3` would compute `phi_start` from the initial position of `bar3.obj2` and `bar2.obj2`. There might

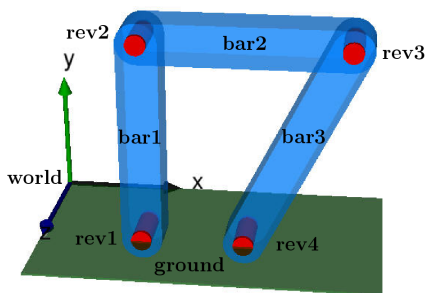


Figure 8. Four-bar mechanism with different link lengths.

be situations in which it is not as simple as in Figure 7 to define a consistent initial configuration. In such cases, Modia3D provides functions to determine kinematic quantities in the *initial* configuration and utilizes them in later constructor calls. For example, assume that the bars of a four-bar mechanism do not all have the same lengths as in Figure 8. The corresponding assembly object can be defined by using functions that compute geometric properties in the initial configuration.

```
79 @assembly Fourbar2(;Lx=0.1,Ly=Lx/5) begin
80   ...
81   bar1 = Bar(Lx=Lx,Ly=Ly)
82   bar2 = Bar(Lx=Lx,Ly=Ly)
83   rev1 = Revolute(pos1,bar1.obj1,
84                 phi_start=pi/2)
85   rev2 = Revolute(bar1.obj2,bar2.obj1,
86                 phi_start=-pi/2)
87   L3   = distance(pos2,bar2.obj2)
88   phi30 = planarRotationAngle(pos2,
89                             bar2.obj2)
90   bar3 = Bar(Lx=L3)
91   rev3 = Revolute(bar3.obj2,bar2.obj2,
92                 phi_start=NaN)
93   rev4 = Revolute(pos2,bar3.obj1,
94                 phi_start=phi30)
95   ...
96 end
```

First, `bar1`, `bar2` (lines 81 - 82) are defined with the `Bar` assembly (lines 49 - 55) and as well as their connection with `revolute` joints `rev1`, `rev2` (lines 83 - 86). As a result, the initial positions of `bar1`, `bar2`, as well as `pos2` (line 61) on the ground are known. In a second step, the distance `L3` between the origin of `pos2` and the origin of `bar2.obj2` is computed (line 87). If `bar3` (line 87) is placed between these two objects, it must have `Lx=L3`. Furthermore, the angle `phi30` (line 88) between the x-axis of `pos2` and the position vector from the origin of `pos2` to the origin of `bar2.obj2` is computed and used as start angle for `rev4`.

Note, the result is similar to a system that is defined by a parameterized CAD system: Whenever `Fourbar2` is instantiated with different arguments (e.g. `Lx=0.5` or `Lx=10.1`), *consistent initial configurations* of the mechanism are constructed always.

4 Actuator Objects

The main purpose of Modia3D is to model the 3D-part of a system. All other parts of a system model shall be defined with the equation-based modeling language Modia. Modia3D and Modia shall be combined in the following ways:

1. using Modia models in Modia3D (e.g. a Modia actuator model that drives a Modia3D revolute joint),
2. using Modia3D models in Modia,
3. transforming Modia3D models to Modia equations (to be used, e.g. in embedded systems), and
4. defining force elements directly with simple Julia macros, mainly to develop the interface to Modia (but without actually using Modia).

Currently, only item 4 has been implemented by providing the Julia macros `@signal` and `@forceElement`. The usage of these macros is sketched below with two examples:

To move the generalized coordinate of a joint kinematically, a `@signal` macro with one output signal is defined:

```
97 @signal Sine(;y_off=0.5,w=1.0,A=1.0) begin
98   y = RealScalar(causality=Output)
99 end

100 function computeSignal(sine::Sine,sim)
101   sine.y.value = sine.y_off +
102                 sine.A*sin(sine.w*sim.time)
103 end
```

Here, one output variable `y` (line 98) is declared as `RealScalar` and it is computed in Julia function `computeSignal(sine,sim)` (line 100). All parameters that are defined in the header declaration (line 97) as well as all variables of the `SimulationState` `sim`, e.g. `sim.time`, `sim.startTime`, can be used for computing the signal (lines 100 - 103). The new type `Sine` can be used in an assembly component e.g. to drive one joint of the four-bar mechanism (Figure 7).

```
104 @assembly MoveFourbar(;Lx=0.1) begin
105   fourbar = Fourbar(Lx=Lx)
106   sine    = Sine(A=0.5,w=2.0)
107   flange  = SignalToFlangeAngle(sine.y)
108   Modia3D.connect(flange, fourbar.rev1)
109 end

110 fourbar = MoveFourbar(Lx=1.0)
111 model   = SimulationModel(fourbar,
112                           analysis=KinematicAnalysis)
113 result  = simulate!(model,stopTime=3.0)
```

In `MoveFourbar` an instance of `Sine` is created (line 106). For connecting this instance with a revolute joint, a converter from pure signals into a rotational flange is needed (line 107). Here, `sine.y` (line 107) is associated with `flange.phi`. The `connect(...)` statement (line 108) copies the corresponding variables from `flange.phi` to `fourbar.rev1.phi`.

Function `SimulationModel(..)` (lines 111 - 112) generates a *simulation model* that can then be simulated with the generic `simulate!(..)` (line 113) function. Since option `analysis=KinematicAnalysis` is defined, the *simulation model* computes the positions of all frames, but no velocities or accelerations and no forces or torques are calculated. The kinematic simulation is done by evaluating the assembly on a regular grid from `time=0.0` up to `time=3.0`. At every time instant of this grid, all `computeSignal(..)` functions of each assembly component are called. This results in the *kinematic simulation* of the four-bar mechanism. Note, since there is a kinematic loop, nonlinear algebraic equations are solved by the `simulate!(..)` function.

The next example shows how a P-PI cascade controller can be defined that drives a rotational flange of a Modia3D assembly:

```

114 @forceElement Controller(;k1=10.0,k2=10.0,
115     T2=0.01,freqHz=0.5,A=1.0) begin
116     PI_x = RealScalar(...)
117     PI_derx = RealScalar(...)
118     sine_y = RealScalar(...)
119     phi = RealScalar(causality=Input)
120     w = RealScalar(causality=Input)
121     tau = RealScalar(causality=Output)
122 end

123 function computeTorque(c::Controller,sim)
124     c.sine_y.value = c.A*
125         sin(2*pi*c.freqHz*sim.time)
126     gain_y = c.k1*(c.sine_y.value -
127         c.phi.value)
128     PI_u = gain_y - c.w.value
129     c.PI_derx.value = PI_u/c.T2
130     c.tau.value = c.k2*(c.PI_x.value + PI_u)
131 end

```

The Controller model uses the angle `phi` and angular velocity `w` as inputs (lines 119 - 120) to compute the driving torque `tau` (line 121) as output. This is performed with a P-PI cascade controller where a sine is used as a reference angle. Here, all parameters have to be defined in a `@forceElement` model, and can be used for computing the driving torque with function `computeTorque(c,sim)` (line 123). This function (lines 123 - 131) takes `c` as an instance of `Controller` and `sim` as an instance of `SimulationState` as input values.

```

132 @assembly MoveFourbar2(;Lx=0.1) begin
133     fourbar = Fourbar(Lx=Lx)
134     c = Controller()
135     flange = AdaptorForceElementToFlange(
136         phi=c.phi, w=c.w, tau=c.tau)
137     Modia3D.connect(flange, fourbar.rev1)
138 end

139 fourbar2 = MoveFourbar2(Lx=1.0)
140 model = SimulationModel(fourbar2)
141 result = simulate!(model,stopTime=3.0)

```

In `MoveFourbar2` an instance of `Controller` is created (line 134). For connecting this instance with a revolute joint (line 137), an adaptor between a force element and

a flange is needed. This is done with function `AdaptorForceElementToFlange(..)` (lines 135 - 136) that uses keywords `phi`, `w`, `a`, and `tau` (see line 136) to associate the controller signals with the corresponding flange variables. Constructor function `SimulationModel(..)` generates a *simulation model*. Since no keyword argument is provided, the default `analysis=DynamicAnalysis` is used. Function `simulate!(..)` (line 141) performs a dynamic simulation of the simulation model `model`. At every time instant, all `computeTorque(..)` functions of each assembly component are executed.

5 Prototype Implementation

In this section some details about the implementation of the Modia3D prototype are given.

5.1 Handler Objects

Independent *handler objects* are responsible for the various computations that have to be carried out. In a first step, the components for the handler objects are identified.

When instantiating an assembly object, the parent-child-relationships between the `Object3Ds` are updated and stored in them. For example, when instantiating a `Bar` assembly (lines 49 - 55), `obj0` is the parent of `obj1`. However, when connecting `bar2.obj1` to `bar1.obj2` with a revolute joint `rev2` (lines 68 - 69), then the parent-child-relationship is updated, so that `Object3D bar1.obj2` becomes the parent of `bar2.obj1`, and `bar2.obj1` becomes the parent of `bar2.obj0`.

During the update process, kinematic loops are also identified. For example the revolute joint `rev4` (lines 72 - 73) introduces a constraint between two `Object3Ds` that are connected in a tree-structure having the same root-`object3D`. Joints which close a loop are just referenced in the corresponding `Object3Ds`, without changing the parent-child-relationship of the `Object3Ds`. The first `Object3D` in the top-most assembly that is not defined with respect to another `Object3D`, is treated as the *world-object3D*. Due to this approach, a tree of connected `Object3Ds` is constructed having the *world-object3D* as its root. As an inspiration the open-source Javascript library *Three.js*¹⁹, was used, to design a similar tree. The Modia3D `Object3D` data structure is hereby similar to the *Three.js* base class `Object3D`.

In a second step, the kinematic loops are analyzed. Currently, the joints that close a kinematic loop are treated as cut-joints. Hereby, the corresponding kinematic loop is analyzed and if the loop is planar, a reduced set of equations are used for the cut-joint. It is planned to significantly improve this phase by analytically solving a large class of loops with the technique described in (Otter et al., 2003) that is used in the Modelica Standard Library (`MultiBody.Joints.Assemblies`) and is based on the more general *characteristic pair of joints* method

¹⁹<https://threejs.org/>: "Lightweight cross-browser JavaScript library/API used to create and display animated 3D computer graphics on a Web browser".

(Woernle, 1988; Hiller and Woernle, 1987) where a kinematic loop is cut at two joints.

In a third step, the constructed tree is traversed and the handler objects are created with the help of the utility functions of section 2.4.

Collision Handler: All Object3Ds where the function call `canCollide(object3D)` returns true are reported to the collision handler. More details are given in section 5.2.

Renderer Handler: All Object3Ds where the function call `isVisible(object3D, renderer)` returns true are collected in a vector of Object3Ds and this vector is reported to the renderer handler. At every communication point of a simulation, the specific renderer functions are called to visualize the objects associated with the Object3Ds in this vector.

Multibody Handler: All Object3Ds are collected together, in depth-first order, in one vector starting from the world-object3D. During simulation, this vector of Object3Ds is traversed forth and back to compute the needed quantities. Additionally, in a second vector the cut-joints are stored.

The multibody handler has currently two modes: In the *kinematic* mode it computes the positions of all Object3Ds and the generalized coordinates of all joints. This is useful to just analyze the mechanism and visualize it to determine whether it is correctly assembled and kinematically moves in the expected way. In the *dynamic* mode a DAE (Differential-Algebraic-Equation) system of the following form is generated:

$$\begin{aligned} \mathbf{0} &= \begin{cases} \mathbf{f}_d(\dot{\mathbf{x}}, \mathbf{x}, t, z_i > 0) \\ \mathbf{f}_c(\mathbf{x}, t, z_i > 0) \end{cases} & (a) & \quad \mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}_d}{\partial \dot{\mathbf{x}}} \\ \frac{\partial \mathbf{f}_d}{\partial \mathbf{x}} \end{bmatrix} \text{ is reg.} & (c) \\ \mathbf{z} &= \mathbf{f}_z(\mathbf{x}, t) & (b) & \end{aligned} \quad (1)$$

where $\mathbf{x} = \mathbf{x}(t)$ and the Jacobian (1c) is regular. Therefore (1a) is an index 1 DAE. (1b) defines zero-crossing functions $\mathbf{z}(t)$. Whenever a $z_i(t)$ crosses zero the integration is halted, functions $\mathbf{f}_d, \mathbf{f}_c$ (1a) might be changed (for example by providing elastic material laws at a contact) and afterwards integration is restarted. The transformation of a multibody system with kinematic loops to this form is sketched in (Otter and Elmquist, 2017). The DAE is solved with Sundials IDA (Hindmarsh et al., 2005, 2015) that uses a variable-step, variable-order BDF-integration method.

The transformation to equations (1) is performed in a configurable way: All variables appearing in equation system (1) must be declared as instances of `RealScalar` or `RealArray`. These types contain all the attributes of the `ScalarVariable` type of the FMI 2.0 standard²⁰ (Blochwitz et al., 2012), as well as some additional attributes to identify the type of the variable with respect to

²⁰<https://fmi-standard.org/>

the variable categories introduced in (Otter and Elmquist, 2017). The multibody handler traverses all assembly objects (including actuator objects) and extracts the information about the variable objects. For example, a `RealScalar` variable `phi` is declared in a revolute joint. The corresponding constructor call defines that `phi` shall be part of vector \mathbf{x} . Whenever the integrator requires a model evaluation, all elements of vector \mathbf{x} are copied to the corresponding variable definitions. Afterwards, the multibody handler computes the residues, which are also defined to be variables, and copies the values of the residue variables back to the residue vector used by the integrator.

5.2 Collision Handling

Collision detection in Modia3D is based on the MPR (Minkowski Portal Refinement) algorithm (Snethen, 2008), which computes the shortest penetration depth of two convex shapes. The MPR-algorithm is much simpler to implement and has less numerical problems than the often used GJK/EPA-standard algorithms (Gilbert et al., 1988; Bergen, 2003), because it only works with triangles and not with tetrahedrons.

DAE (1) generated by Modia3D is solved with a variable-step integrator. Variable-step integrators are sensitive to drastic changes of the DAE, as in the case of collisions. To speed up the simulation and to improve the robustness of the integration, Modia3D uses the distances between convex shapes as zero-crossing functions $z_i(t)$ (1b). In the original version of the MPR-algorithm (Snethen, 2008) only penetration depths are determined. In Modia3D improvements of the MPR-algorithm are utilized that have been proposed in (Kenwright, 2015; Neumayr and Otter, 2017), in particular to compute the distances of shapes that are not in contact, treating special collision situations properly and introducing a new termination condition to speed up the algorithm in some situations.

In Modia3D collision handling of n potentially colliding shapes is performed in the following (mostly standard) way:

1. *Broad Phase*
The shapes are approximated by bounding volumes where potential collisions can be very cheaply determined resulting in $O(n^2)$ cheap tests. When using special data structures (such as octrees or kd-trees), it is possible to reduce the number of cheap tests to $O(n \log(n))$.
2. *Narrow Phase*
For the potentially colliding shape pairs as identified in the broad phase, the signed distances are computed with the improved MPR-algorithm (Neumayr and Otter, 2017).
3. *Response Calculation*
If two shapes are penetrated, a force and/or torque is applied at the contact point, such as a spring - damper

force element, depending on the penetration depth (section 2.3).

The MPR-algorithm computes the contact points C_A, C_B , the Euclidean distance δ if shapes are not in contact, and otherwise the penetration depth δ (Figure 9). The distance

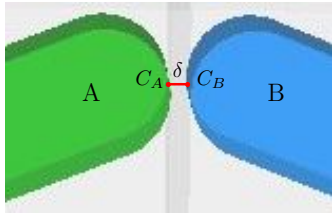


Figure 9. Shapes A, B are not in contact.

δ calculated by the MPR-algorithm is used as zero-crossing function z_i for the integrator. This means it detects the transition between penetration and non-penetration of a shape pair. A brute force method for the integrator would be to use the distances between any two shapes as zero-crossing function, resulting in an $O(n^2)$ number of zero-crossing functions. Since the number of crossing functions would grow quadratically with the number of collision objects, the maximum number of zero-crossing functions is bounded by $n_{z,max}$, which defines the maximum number of objects that can be in contact at the same time instant. This number can be adapted by the user. If more shapes get in contact, the simulation is currently halted with an error (alternatively, the simulation could be halted and could be restarted with an enlarged z vector). The zero-crossing functions are computed with the following scheme (for more details, see (Neumayr and Otter, 2017)):

- The function `selectContactPairs!(..)` is called before every integrator step.
 - Execution of broad and narrow phase.
 - Selection and ordering of $n_z \leq n_{z,max}$ shape pairs according to their distances.
- The function `getDistances!(..)` is called whenever the integrator requests a new zero-crossing function evaluation.
 - Execution of broad and narrow phase.
 - Storing the distances of the contact pairs in z that have been selected in the last call of `selectContactPairs!(..)` and checking that none of the remaining distances is negative.

The broad phase in Modia3D uses *AABBs* (= Axis Aligned Bounding Boxes) (see e.g. (Bergen, 2003)). Each *AABB* approximates one shape and only if the *AABB*'s are intersecting, the distance between these two possibly colliding shape pairs is calculated in the narrow phase. In the narrow phase, *support points* (Bergen, 2003; Snethen, 2008) are computed. A support point is a point on a shape which is farthest away in the search direction e and is computed as

```

142 supportPoint (geo, r_abs, R_abs, e) =
143   r_abs + R_abs' * (centroid(geo)
144     + supportPoint_ref (geo, R_abs * e))

```

where `supportPoint_ref(..)` is the shape-specific function to compute a support point in the reference coordinate system of the shape.

The *AABB* of a shape is calculated by calling the `supportPoint_ref` function specialized for one axis $i = 1, 2, 3$ in a particular axis direction $dir = -1, 1$.

```

145 supportPoint_i (geo, r_abs, R_abs, i, dir) =
146   r_abs[i] + R_abs[:, i]' * (centroid(geo)
147     + supportPoint_ref (geo, dir * R_abs[:, i]))

```

Therefore, no shape specific *AABB* function is needed. The *best fitting* *AABB*'s are not useful when zero-crossing functions shall be computed, because if some surfaces or edges of a shape are also parallel to an axis, and these shapes would incidentally collide, they are already penetrating each other (see Figure 10). Therefore, it will

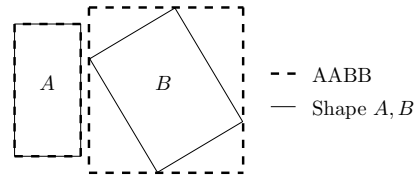


Figure 10. Best fitting *AABB*'s.

not be possible for the variable-step integrator to detect the transition between penetration and non-penetration. Hence to avoid such scenarios, each edge length of the best fitting *AABB* gets enlarged by a specific factor of the longest edge length. In Figure 11 there are four shapes A_1, A_2, B_1, B_2 and each have its *AABB*'s shown as a grey box. Collision handling for shape pairs is switched off,

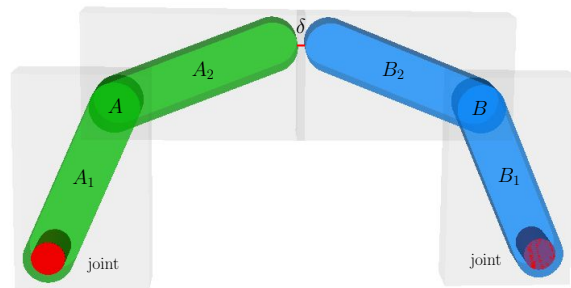


Figure 11. Two rigidly attached shapes with *AABB*'s.

when shapes are rigidly connected to each other, or when shapes are connected by a joint and the joint-specific option `canCollide` is set to `false` (= the default setting). This reduces the amount of possible collision pairs before the broad phase is executed. For example, shapes A_1, A_2 in Figure 12 are rigidly connected. So A_1 cannot collide with A_2 , but both shapes can still collide with all other shapes. In Figure 12, the red cylinders characterize revolute joints. Therefore, not 6 but only 2 shape pairs ($A_1 - C$ and $A_2 - C$)

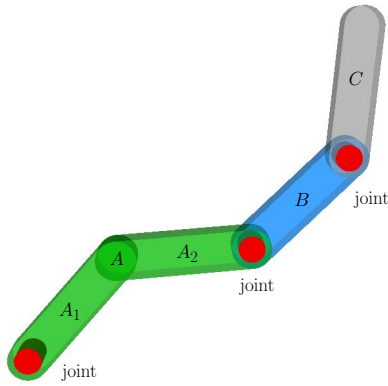


Figure 12. Rigidly attached shapes and joints.

are checked in the broad phase. In Figure 11, there are two rigidly attached shapes A , that consists of A_1, A_2 , and B , that consists of B_1, B_2 . The joints, which connects them to the ground are visualized with red cylinders. Without any assumptions, there would be 6 possible pairs to check in the broad phase. But by pre-processing the structure of the computational tree, it is reduced to 4 pairs, that have to be looked at in the broad phase whether the AABB's are intersecting. Here only for one pair $A_2 - B_2$ the narrow phase (MPR-algorithm) has to be executed.

5.3 Compilation Time

All equations to compute the movement of Object3Ds and joints are implemented in Julia functions that can be compiled once and then just called for the actual model. Therefore, basically the same compiled code is used for any model, independent of its size. Only the `@assembly` code that describes which Object3Ds, joints etc. are used and how they are connected and parameterized is compiled for an actual model. But this code part is very small as compared to all the other equations. Hence, compiling a Modia3D model should be fast and nearly independent of model size. In an equation-based modeling system the equations of every instance need to be symbolically processed and translated. Therefore, the translation time grows with model size. To clarify this behavior, the following experiment was carried out:

The mechanical part of the 6 degree of freedom r3-robot present in the Modelica Standard library²¹ was used in a comparison test. In Table 1 the translation/compilation time (= time from requiring to simulate the model, until the simulation starts) of OpenModelica (1.13.0 nightly build) and of a commercial Modelica tool were compared with the compilation time of the corresponding Modia3D r3-robot model.

As expected, the simulation of the Modia3D model starts nearly immediately even for large models, whereas a waiting time is present for a Modelica model before simulation starts and this can be significant for large models.

²¹Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3-Components.MechanicalStructure

| | Number of robots | | | |
|--------------------------|------------------|-------|--------|-------|
| | 1 | 10 | 50 | 100 |
| OpenModelica | 17 s | 194 s | 3600 s | — |
| commercial Modelica tool | 5 s | 20 s | 80 s | 170 s |
| Modia3D | 0.3 s | 0.4 s | 0.5 s | 0.6 s |

Table 1. Translation/compilation time for 1...100 robots (= 6...600 degrees of freedom) on a standard notebook.

6 Relation to other Work

*Multibody systems software*²² is designed to simulate mechanical systems, often in offline simulations. A large number of multibody codes exist such as ADAMS, RecurDyn, SIMPACK and many others²³. Typically, specialized integration methods based on variable-step integrators are used. Furthermore, it is standard to support mechanisms with *kinematic loops* in a *numerically sound way*.

Modia3D has these features of a multibody program. However, the architecture of a typical multibody program is centered around rigid or flexible bodies where points on the body are specially marked and then objects (joints, forces, visual elements, etc.) are connected to these markers. Modia3D instead is centered around component-based design where optional components are associated to coordinate systems. The advantage is that models with many variants can be much more flexibly configured without code-duplication. For example, in the Modelica Multi-Body library there are many parts, such as `BodyShape`, `BodyBox` etc. and every part defines a fixed variant (e.g. `BodyBox` defines a rigid body and a visual shape from a geometric box). Obviously, the number of manageable variants is limited by this design and similar code fragments are used at many places (e.g. to locate a shape object relatively to the part reference frame). Furthermore, it is planned to extend Modia3D also in non-mechanical domains (such as optionally adding heat transfer to a solid) which is straightforward with the component-based design. On the other hand, Modia3D is an experimental prototype and features are missing that are available in widely used multibody codes and are important in industrial applications.

*Game engines*²⁴, such as Unity or Unreal engine, are used to develop games. Typically, fixed-step integrators are used in game engines, collision handling is a key element and simulation of mechanisms with kinematic loops is either not or only approximately supported. Modia3D supports collision handling in a similar way as in a game engine (currently, only elastic response calculation is supported, but it is planned to add optional impulse-based response computations). Due to the component-based design it is easy to configure the geometries that shall be treated in the collision handling (= by providing a contact material). There had been several attempts to support collision handling in Modelica, such as (Otter et al., 2005; Hofmann

²²https://en.wikipedia.org/wiki/Multibody_system

²³see e.g.: <https://www.ifto-mm-multibody.org/software>

²⁴https://en.wikipedia.org/wiki/Game_engine

et al., 2014; Elmqvist et al., 2015b; Bardaro et al., 2017). These approaches use external C or C++ programs for the collision handling and interface these programs to Modelica. The drawback is that a close integration into a model is hard. For example, *new* parts are provided that support collision handling (existing parts, such as `BodyBox` do not get this feature), and the same geometry is present three times: For collision handling, for animation, and for computing the rigid body properties. In Modia3D, a geometry, such as a box, is only present once. In the constructor call it is defined whether mass properties are computed from the geometry, whether the geometry is shown in the animation or whether it is utilized in collision handling, or any variant of these options.

7 Conclusion

In this article a new technique is proposed to improve modeling of 3D systems for a modeling language. Ingredients from different communities are used: The basic architecture is taken from game engines, in particular to use component-based 3D modeling to achieve a very flexible way to build-up 3D systems, to model collisions and to use various handlers for the different computational tasks. Kinematic and dynamic simulation is performed with multibody algorithms, in particular to simulate systems with kinematic loops, and by utilizing variable-step integrators with zero-crossing functions. Constructing consistent initial configurations is performed by using ideas from parameterized CAD systems. The hierarchical modeling and naming of sub-components follows the Modelica/Modia approach. The equation-based modeling language Modia shall be used to provide dynamic models from other domains, e.g. as actuators to drive a joint. On the other hand, it is planned that Modia3D models can be utilized as components in a Modia model. As a résumé it can be noted that the proposed approach seems to considerably improve the 3D modeling features of an equation-based language and could therefore be used as one building block of the next Modelica generation.

Modia3D is still an early prototype and several important parts are under development, especially the integration with Modia is missing. Furthermore, the code was currently mainly developed for its functionality and not yet tuned for efficiency. For these reasons, benchmarks about the simulation efficiency have not yet been performed, especially also not for large models (e.g. sparse matrix handling in the simulation engine was tested, but is not yet available in the publicly available prototype).

References

G. Bardaro, L. Bascetta, F. Casella, and M. Matteucci. Using Modelica for advanced Multi-Body modelling in 3D graphical robotic simulators. In J. Kofranek and F. Casella, editors, *Proc. of the 12th International Modelica Conference*. LiU Electronic Press, May 2017. URL <http://www.ep.liu.se/ecp/132/097/ecp17132887.pdf>.

T. Bellmann. Interactive Simulations and advanced Visualization with Modelica. In Francesco Casella, editor, *Proc. of the 7th International Modelica Conference*. LiU Electronic Press, Sept. 2009. URL <http://www.ep.liu.se/ecp/043/062/ecp09430056.pdf>.

G.v.d. Bergen. *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann Publishers, 2003.

J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59(1):65–98, 2017.

T. Blochwitz, M. Otter, J. Akesson, M. Arnold, C. Clauß, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel. The Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In Martin Otter and Dirk Zimmer, editors, *Proc. of the 9th International Modelica Conference*. LiU Electronic Press, Sept. 2012. URL <http://www.ep.liu.se/ecp/076/017/ecp12076017.pdf>.

H. Elmqvist, S. E. Matsson, and C. Chapuis. Redundancies in Multibody Systems and Automatic Coupling of CATIA and Modelica. In *Proceedings of the 7th International Modelica Conference; Como; Italy; 20-22 September 2009*, pages 551–560. Linköping University Electronic Press, 2009. URL <http://www.ep.liu.se/ecp/043/063/ecp09430113.pdf>.

H. Elmqvist, A. D. Baldwin, and S. Dahlberg. 3D Schematics of Modelica Models and Gamification. In Peter Fritzson and Hilding Elmqvist, editors, *Proc. of the 11th International Modelica Conference*. LiU Electronic Press, Sept. 2015a. URL <http://www.ep.liu.se/ecp/118/057/ecp15118527.pdf>.

H. Elmqvist, A. Goteman, V. Roxling, and T. Ghandriz. Generic Modelica Framework for MultiBody Contacts and Discrete Element Method. In Peter Fritzson and Hilding Elmqvist, editors, *Proc. of the 11th International Modelica Conference*. LiU Electronic Press, Sept. 2015b. URL <http://www.ep.liu.se/ecp/118/046/ecp15118427.pdf>.

H. Elmqvist, T. Henningsson, and M. Otter. Systems Modeling and Programming in a Unified Environment based on Julia. In *Proc. of ISO/FAO Conference*. Springer, Oct. 2016. doi:10.1007/978-3-319-47169-3_15.

H. Elmqvist, T. Henningsson, and M. Otter. Innovations for Future Modelica. In J. Kofranek and F. Casella, editors, *Proc. of the 12th International Modelica Conference*. LiU Electronic Press, May 2017. URL <http://www.ep.liu.se/ecp/132/076/ecp17132693.pdf>.

E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, 1988. URL <https://graphics.stanford.edu/courses/cs448b-00-winter/papers/gilbert.pdf>.

M. Hellerer, T. Bellmann, and F. Schlegel. The DLR Visualization Library - Recent development and applications. In Hubertus Tummescheit and Karl-Erik Arzen, editors, *Proc.*

- of the 10th International Modelica Conference. LiU Electronic Press, March 2014. URL <http://www.ep.liu.se/ecp/096/094/ecp14096094.pdf>.
- M. Hiller and C. Woernle. A Systematic Approach for Solving the Inverse Kinematic Problem of Robot Manipulators. In *Proceedings 7th World Congress Th. Mach. Mech.*, 1987.
- 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(3):363–396, September 2005.
- A.C. Hindmarsh, R. Serban, and A. Collier. User Documentation for IDA v2.8.2. Technical Report UCRL-SM-208112, Lawrence Livermore National Laboratory, 2015.
- A. Hofmann, L. Mikelsons, I. Gubsch, and C. Schubert. Simulating Collisions within the Modelica MultiBody Library. In Hubertus Tummescheit and Karl-Erik Arzen, editors, *Proc. of the 10th International Modelica Conference*. LiU Electronic Press, March 2014. URL <http://www.ep.liu.se/ecp/096/099/ecp14096099.pdf>.
- B. Kenwright. Generic Convex Collision Detection using Support Mapping. Technical report, 2015. URL <https://www.semanticscholar.org/paper/Generic-Convex-Collision-Detection-using-Support-Kenwright/4f0f2d95375db7cfdbfaa345847418789d8cb970>.
- A. Neumayr and M. Otter. Collision Handling with Variable-step Integrators. In *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, EOOLT'17, pages 9–18. ACM, 2017.
- R. Nystrom. *Game Programming Patterns*. Genever Benning, 2014. URL <http://gameprogrammingpatterns.com/>.
- M. Otter and H. Elmqvist. Transformation of Differential Algebraic Array Equations to Index One Form. In J. Kofranek and F. Casella, editors, *Proc. of the 12th International Modelica Conference*, May 2017. URL <http://www.ep.liu.se/ecp/132/064/ecp17132565.pdf>.
- M. Otter, H. Elmqvist, and S. E. Mattsson. The New Modelica MultiBody Library. In P. Fritzson, editor, *Proc. of the 3rd International Modelica Conference*, Nov. 2003. URL https://www.modelica.org/events/Conference2003/papers/h37_Otter_multibody.pdf.
- M. Otter, H. Elmqvist, and J. Diaz Lopez. Collision Handling for the Modelica MultiBody Library. In Gerhard Schmitz, editor, *Proc. of the 4th International Modelica Conference*, March 2005. URL https://modelica.org/events/Conference2005/online_proceedings/Session1/Session1a4.pdf.
- G. Snethen. Xenocollide: Complex collision made simple. In Scott Jacobs, editor, *Game Programming Gems 7*, pages 165–178. Charles River Media, 2008.
- C. Woernle. *Ein systematisches Verfahren zur Aufstellung der geometrischen Schliessbedingungen in kinematischen Schleifen mit Anwendung bei der Rückwärtstransformation für Industrieroboter*. Fortschrittsberichte VDI. Reihe 18, ISSN 0178-9457, 1988.

The Deployable Structures Library

Cory Rupp Laura Schweizer

ATA Engineering, Inc., USA, {cory.rupp, laura.schweizer}@ata-e.com

Abstract

Deployable structures are an enabling technology for many space- and ground-based structures and vehicles. Analysis of deployment mechanisms and structural dynamic responses in early design phases is key to ensuring deployment reliability and overall structural integrity. In this paper, a Modelica library is presented that provides a number of building blocks to enable and ease the development of models of deployable structures. Several examples using the library are presented that would be difficult or impossible to model using other technologies.

Keywords: Modelica, deployable structures, flexible structures, spacecraft, solar array

1 Introduction

Accurate modeling of multibody kinematics and dynamics is critical for design and analysis of deployable structures and mechanisms. This is particularly true for expensive, highly engineered space-based structures such as solar arrays, antennas, and booms where deployment failure results in mission failure. Not only is deployment the number one risk to these systems, they also must meet stringent mass and stiffness requirements that make structural dynamics responses an important consideration in preliminary design phases.

Existing software tools for performing multibody simulations are often domain-specific and limited in extensibility. The primary tool used in the aerospace industry is ADAMS, a proprietary software package provided by MSC. While ADAMS is primarily geared toward multibody analysis, it can be extended by integration with Simulink and some other domain-specific tools. Although possible, implementing multiphysics effects or specialized mechanisms is far from trivial and largely beyond the intended scope of the ADAMS toolset. On the other hand, Modelica, specifically the MultiBody library of the Modelica Standard Library (MSL), has far more freedom for the user to add new capabilities for analysis of deployable structures. Even so, the MSL is limited in the range of structures it can model.

This paper presents a new Modelica library, the Deployable Structures Library (DSL), which provides specialized structural and mechanism components that are useful for modeling deployable structures. This library is largely an extension of the MSL MultiBody

library to expand its applicability. In addition to new modeling capabilities, built into the library is a modeling workflow that more closely follows the typical structural engineering design and analysis process. With the library being tailored to the engineering process, it is hoped that structural engineers will be able to more readily adopt the library and Modelica as a modeling tool in general.

2 The Deployable Structures Library

The Deployable Structures Library contains a mix of new modeling capabilities and reformulations or extensions of existing MSL blocks to provide a toolset for structural design and analysis engineers. The library enables the analysis of many space-based deployable structures and other difficult-to-analyze spacecraft components, as well as ground-based structures. Many of these systems include complex, one-off mechanisms to actuate their movement. As such, it is imperative to have a modeling technology such as Modelica that is extensible and can be used to model the mechanism's behavior at a fundamental level. The goal of the library is to make building models of these mechanisms easier through use of a set of common building blocks.

The DSL is organized into several packages, somewhat mimicking the organization of the MSL MultiBody library. At the top level are the Examples, Interfaces, Math, Parts, Properties, Utilities, and Visualization packages, as shown in Figure 1. The most relevant of these are described in the following sections.

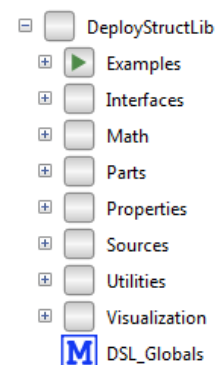


Figure 1. Top-level view of the Deployable Structures Library.

2.1 DSL_Globals

At the top level of the library is the *DSL_Globals* block. This block is used as a Modelica inner model, holding

global parameters that enable specialized structural analyses, including steady-state and quasi-static analysis, to be performed. Components in the Parts package reference this block with Modelica's outer construct to change their set of equations and model different behavior. For example, the quasiStaticFactor parameter will scale the mass values of all DSL components, typically using a construct such as

```
parameter SI.Mass mass =
  if DSglb.quasiStatic then
    DSglb.quasiStaticFactor*rho*xprop.A*L
  else
    rho*xprop.A*L "Beam mass";
```

What this does is reduce the inertia consistently across the entire simulation, thereby reducing the problem to a quasi-static problem. It is equivalent to pushing the mass matrix toward zero (i.e., $M \rightarrow 0$) in the standard dynamics equation

$$M\ddot{u} + C\dot{u} + Ku = f \quad (1)$$

leading to (assuming also $C \rightarrow 0$) the static equation

$$Ku = f \quad (2)$$

With this change, time-varying static structural responses can be evaluated using a transient solver in a Modelica vendor-independent manner.

2.2 Properties

To help reduce the barrier to adoption and implementation of Modelica for engineers, much of the library has been designed with the structural engineering workflow in mind. As an example, typical structural engineering analysis software is organized with material and structural properties defined separately from the actual structure model. As such, a single property only needs to be defined once, thereby preventing model bloat and modeling error. This concept is implemented in the Properties package in a natural way through Modelica record parameters, which are used by structural elements in the library.

Within the Properties package are subpackages of records that can be used for material property, beam cross section, and cloth property definitions (Figure 2). When a model is created, these records can be implemented at the top level as parameters that are passed down through the model to individual Deployable Structures Library components. This workflow is similar to modeling procedures used by other structural engineering tools such as finite element software packages where elements are given a material property identifier that corresponds to a single property definition. With such a process, fewer mistakes are made because there are fewer opportunities for error. This implementation in Modelica has the added benefit that design studies with material, beam cross section, or cloth properties can be performed easily.

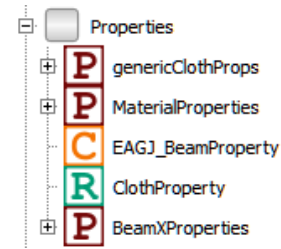


Figure 2. The Properties subpackage of the DSL.

2.3 Parts

The Parts package contains a plethora of new and extended components and mechanisms that can be used to model deployable structures (Figure 3). The most relevant and broadly applicable of these are discussed in the following sections.

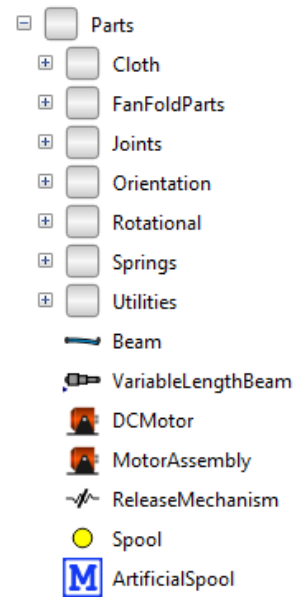


Figure 3. The Parts subpackage of the DSL.

2.3.1 Beams

It is virtually impossible to model deployable structures without some kind of beam model. While the MultiBody package in the MSL contains rigid components that can be used to model rigid beams (namely, the *BodyBox* block), changing the beam cross section requires explicit (and error-prone) parameterization, and there is no means to accurately model beam flexibility. Most deployable structures, especially space-based structures, contain lightweight and thin beam members, making it absolutely necessary for beam cross sections to be designed and beam flexibility included in models, as beam deformation dramatically affects overall dynamic motion and response.

To make it easier to model thin structural members, the Parts package within the DSL contains a *Beam* block that acts as a wrapper for both rigid and flexible beam models. The block requires appropriate material and cross-sectional properties as parameter inputs, which are defined through blocks in the DSL.Properties

subpackage. The material properties are provided to the *Beam* block through a *MaterialProperties* record, which allows the user to define independent materials and pass them to individual beams. The beam cross-sectional properties can be passed to the *Beam* block either by setting the dimensions of the cross section in a *BeamXProperties* record and passing that record to the beam or by setting combined properties such as the bending rigidity EI and the torsional rigidity GJ in an *EAGJ_BeamProperty* block and passing these to the beam. In the case of the *BeamXProperties* records, several predefined cross-sectional shapes are provided in which cross-sectional moments of inertia are automatically computed when the model is built. Beam materials are currently assumed to be isotropic, and cross-sections are assumed to be constant along the length.

Whether a beam is modeled as flexible or rigid is determined by setting the value of the Boolean *rigid* parameter. This parameter selects between the *FlexBeam* and *RigidBeam* blocks described next. The beam model itself is essentially a dummy model layer that implements either a *FlexBeam* or *RigidBeam* depending on the value of the rigid Boolean parameter. The Modelica code for this implementation layer looks like this:

```
parameter Boolean rigid = false;
RigidBeam beamR(L=L, xprop=xprop, ...) if
  rigid;
FlexBeam beamF(L=L, xprop=xprop, ...) if not
  rigid;
equation
if rigid then
  connect (beamR.frame_a, frame_a);
  connect (beamR.frame_b, frame_b);
else
  connect (beamF.frame_a, frame_a);
  connect (beamF.frame_b, frame_b);
end if;
```

There is no requirement that all beams in a model be flexible or all be rigid since the flag is set at the level of the individual beam. However, efficiencies can be gained when creating and debugging a model by first using rigid beams and then later switching to a flexible model through the use of a single top-level parameter tied to all the beam *rigid* parameters.

2.3.2 Flexible Beam

The *FlexBeam* block provides a Bernoulli-Euler model of a beam. The formulation of this block follows that of Schiavo *et al* (2006), with some modifications to fit into the overall scheme of the DSL and without the internal element discretization (i.e., the block consists of a single finite element). In particular, the beam cross-sectional and material properties are passed in as parameter blocks from the `DeployStructLib.Properties` subpackage. While not as general as the DLR FlexibleBodies library (2006) or that described by

Ferretti *et al* (2014), this Modelica-based flexible beam model is intended to provide basic functionality when flexibility is a concern in the analysis of deployable structures.

The key assumption of Bernoulli-Euler beams is that plane sections that are normal to the neutral axis of the beam remain plane and normal to the axis after bending. This implies that transverse shear effects are neglected in forming the stiffness matrix; thus, this block is primarily intended for modeling slender beams.

The mass matrix in the flexible beam can be computed either as a lumped mass matrix, where all mass is lumped into the beam degrees of freedom, or as a consistent mass matrix following typical finite element procedures through the Boolean parameter *useLumpedMassMatrix*. In structural dynamics analysis, the formulation of the mass matrix is often an important consideration for accuracy, and often the lumped mass matrix is used. The lumped mass matrix for the particular finite element formulation used here, however, is singular, as the inertia of the bending degrees of freedom lumps to zero. This leads to singular matrix errors during solving, for which the solution is to add an MSL *Body* block to the beam tip with a small mass value, which in effect adds rotational inertia terms to the beam bending degrees of freedom, thereby rectifying the singular matrix issue. Because this modeling caveat is a nonstandard, non-intuitive procedure, the *useLumpedMassMatrix* parameter is false by default so that general users can use the flexible beam block and need not understand the nuances of finite element mass matrices. Corresponding clarification on the use of the *useLumpedMassMatrix* parameter is provided in the *Beam* block documentation.

Structural damping in the beam is computed using Rayleigh damping; i.e., the damping is a weighted linear combination of the stiffness and mass matrices.

A second flexible beam block, *FlexBeamEAGJ*, is also provided. This block uses the same formulation as the *FlexBeam*, but it allows the user to specify beam properties such as the bending rigidity EI, rather than separately specifying the material and cross-sectional properties. This is useful for correlating models to test data, especially when the beam under consideration has a complicated cross section, is a compound beam, or is built from composite materials.

2.3.3 Rigid Beam

The *RigidBeam* block ignores the stiffness of the beam and assumes complete rigidity. It is based on the *BodyBox* block in the MultiBody library of the MSL but includes inertial calculations from material and cross-section definitions using the same `DSL.Properties` blocks as used in the *FlexBeam* model.

2.3.4 Variable Length Beam

The *VariableLengthBeam* block provides a flexible beam model that can change in length over time and

simultaneously update its stiffness and inertia properties. At each time step, the block recalculates the inertia tensor of the beam, correctly accommodating large changes in beam length. The cross-section of the beam is assumed to be constant, so only the change in length is included when the stiffness and inertia are recalculated. The formulation is the same as that of the *FlexBeam* block but with the addition that the length of the beam is defined by an initial length parameter and a length rate of change input provided by a *VariableLengthSource* block (located in the Utilities subpackage of the library) or a standard MSL source block. The Modelica implementation of the variable length is implemented as

```

parameter Real L_start = 1.0 "Start value
  for L";
Real L(start=L_start, fixed=true) "Beam
  length";
Real dL "dL/dt";
Modelica.Blocks.Interfaces.RealInput
  dL_in;
equation
  dL = der(L);
  dL = dL_in;

```

from which downstream variables such as the beam mass, mass and stiffness matrices, and rigid body inertia matrices are no longer parameters but real variables that vary throughout the simulation.

It is important to note that additional terms attributed to time derivatives of inertia, mass and stiffness matrices, and other typically constant parameters are neglected. As such, this simple (and perhaps simplistic) implementation is restricted to analyses where the change in length of the beam is slow. This rate of change restriction can be generalized by ensuring that the rate of change of structural natural frequencies is much smaller than the natural frequencies themselves (i.e., the system parameters could be described as quasi-static). It can be argued that this assumption is valid for the vast majority of deployable structures, in particular space-based structures, as the deployment process often occurs over the course of several minutes to keep structural loads low. The *VariableLengthBeam* model, however, should be used with care and could potentially be extended to more general cases through implementation of missing terms via derivations such as those provided in Yang *et al* (2017).

The *VariableLengthSource* block used as input to the *VariableLengthBeam* block is primarily for convenience and outputs a user-defined change in length per unit time for a given beam deployment sequence. The rate of change is a parameter, so it is constant throughout the simulation. The user can also set a minimum length and a maximum length so that the beam length stops at a set deployed distance.

2.3.5 Weak Joints

The Joints subpackage contains two “weak” joint models: a *WeakSpherical* joint and a *WeakRevolute* joint. Most joints in the MSL use strong constraints, e.g., the positions of frame_a and frame_b must be exactly equal. Many deployable structures have kinematically complicated systems of joints that may ultimately connect in a kinematic loop. Handling of kinematic loops is a Modelica vendor-dependent operation, so use of a weak formulation overcomes these difficulties on the Modelica side by allowing the user to break the kinematic loop in a specific (and perhaps more desirable) location.

In the weak joint formulation, frame_a and frame_b are essentially connected by springs and dampers, the stiffness and damping of which are parameters that the user sets. The joints use a single value of stiffness and damping for all three translation and rotation directions.

2.3.6 Rotational

The Rotational package of the DSL contains rotational hard stops and rotational locks, both of which are common components in deployable systems.

There are two rotational stop models in the DSL: *RotationalStop* and *DampedRotationalStop*. These two blocks use linear 1D rotational springs to model the stiffness of a hard stop. When the hard stop is not engaged, frame_b of the block is allowed to rotate freely with respect to frame_a, and no torque is generated. When frame_b moves past the given stop angle in the direction opposite free-play, the rotational spring engages and acts to stop the motion of frame_b. The *DampedRotationalStop* block includes a rotational damper in parallel with the rotational spring. Both the damping coefficient and the spring stiffness are set by the user. The rotational stop models do not prevent frame_b bouncing against the stop. That is, if frame_b hits the stop, it can rebound, undamped and unrestricted, in the free-play direction, whether damping is included or not.

The Rotational package also includes two rotational lock models: *RotationalLock* and *DampedRotationalLock*. Both of these blocks monitor the angle of frame_b with respect to frame_a. Once that angle moves past the user-specified locking angle, a linear 1D spring engages to prevent further rotation in the free-play direction. The stiffness of the spring is a parameter set by the user. The *DampedRotationalLock* model includes damping in addition to the spring stiffness.

2.3.7 Springs

The Springs package contains several 3D linear springs that are useful for modeling deployable structures. We highlight several important blocks here.

The translational complement of the *RotationalStop* block is the *BarrierSpring* block. The *BarrierSpring*

block provides a linear spring that acts only in compression. The length of the spring is computed as the distance between `frame_a` and `frame_b` in a user-specified direction that is normal to the plane of the hard-stop. Thus, as the spring length in the direction normal to the hard stop shrinks, the compressive force resisting that motion increases, effectively preventing `frame_b` from colliding with or passing `frame_a`. Since the spring does not act in tension, however, there is no resistance to `frame_b` bouncing against the hard stop in the extensional direction. The *BarrierSpring* block also includes damping; the use of damping in the model can be turned on by setting a nonzero damping coefficient. The spring stiffness, as with the rotational block, is also a user-specified parameter.

Many deployable structures are also tensioned structures, so the *TensionSpring* block is provided for modeling this type of behavior. This block reformulates the MSL *MultiBody.Forces.Spring* block with a *semiLinear* stiffness function to provide stiffness only when the spring is in tension. The *CompressionSpring* block is the compression-only complement to this block.

Deployable structures often contain some mechanism to release the structure from its undeployed configuration and begin the deployment process. The *ReleaseMechanism* block in the Springs package provides a method of modeling the release of the stiffness that holds the structure in place, and it consists of a 3D linear tension spring. When the spring is engaged, it acts to pull `frame_a` and `frame_b` together. A Boolean *released* variable can be set to *true* by the simulation at any time, at which point the mechanism will release, disengaging the spring. To reengage the spring, the *released* flag is set to *false*. Damping can be optionally specified by setting the damping coefficient to a nonzero value, which can be helpful for minimizing chatter in complex deployment mechanisms. Like the spring stiffness, the damping force is only active when the mechanism is not released.

2.3.8 Other Modeling Components

Several other modeling components are provided in the DSL that help in the modeling effort of deployable structures but are not significant enough for detailed description. Among these is a *Spool* block that enables modeling of spooled lanyards or similar gradual release mechanisms, a *GravityRamp* function for use with *MultiBody.World* that aids in quasi-static analysis by slowly ramping up the gravitational acceleration over a period of time, and a series of predefined *Orient* blocks that are simplified versions of the MSL *MultiBody.Parts.FixedRotation* block with preset angles and rotation directions.

2.4 Cloth

To achieve high specific power (the ratio of generating power to mass), space-based solar arrays often utilize

solar blankets in which solar cells are affixed to a lightweight flexible fabric or cloth substrate. Because of their ability to easily fold into a small volume, the solar blankets also provide a high ratio of power to stowed volume, another important metric for spacecraft design. The downside to solar blankets is that they are very flexible and thus require complex structural mechanisms deploy and tension them. Naturally, their flexibility also introduces significant kinematic and dynamic behavior during the deployment process that can potentially damage the spacecraft, the deployment mechanism, or the solar blanket itself. One way this process can go wrong was illustrated when the solar blanket on the International Space Station (ISS) tore during deployment (Wright, 2007).

To model the behavior of solar blankets and other clothlike structures, the DSL provides the *Cloth* package and corresponding modeling block (Figure 4). The *Cloth* block is a high-level building block for creating a discretized mesh of specially formulated membrane finite elements on a defined surface geometry. The membrane elements have displacement degrees of freedom only, so bending moments are ignored in the formulation, which is a suitable assumption for most fabrics. Each membrane element node is connected to a new Modelica *Location* connector consisting of only a 3D position in space and corresponding cut forces for flow variables. The *Location* connector is defined as:

```
connector Location "Location of the
component with one cut-force"
SI.Position r_0[3] "Position vector from
world frame to the connector frame
origin, resolved in world frame";
flow SI.Force f[3] "Cut-force resolved
in world frame"
end Location;
```

The elements in the *Cloth* block are connected to each other by connecting their *Location* interfaces. A lumped mass approach is taken to model the fabric mass distribution whereby point masses with *Location* interfaces are connected to the *Cloth* element nodes.

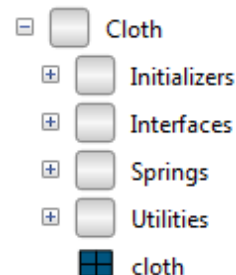


Figure 4. The Cloth package of the DSL.

The formulation of the cloth membrane finite element uses the relative displacement of nodes measured along the element edge as the primary elemental degrees of freedom q . In the Modelica model, this is implemented as

equation

```
d12 = location[1].r_0 - location[2].r_0;
d23 = location[2].r_0 - location[3].r_0;
d31 = location[3].r_0 - location[1].r_0;
q = {Vectors.length(d23)-L[1],
      Vectors.length(d31)-L[2],
      Vectors.length(d12)-L[3]};
```

where d_{12} are relative displacement vectors between nodes i and j and where L_i are the undeformed edge lengths. Elemental strains along an edge, the so-called natural strains ϵ , can be calculated as

$$\epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{bmatrix} = \begin{bmatrix} 1/L_1 & 0 & 0 \\ 0 & 1/L_2 & 0 \\ 0 & 0 & 1/L_3 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = \mathbf{B}_q \mathbf{q} \quad (3)$$

which are merely a transformation of Cartesian strains \mathbf{e} onto the element edges (Fellipa, 2017) assuming a constant strain triangle element, i.e.,

$$\epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \end{bmatrix} = \begin{bmatrix} c_1^2 & s_1^2 & s_1 c_1 \\ c_2^2 & s_2^2 & s_2 c_2 \\ c_3^2 & s_3^2 & s_3 c_3 \end{bmatrix} \begin{bmatrix} e_{xx} \\ e_{yy} \\ 2e_{xy} \end{bmatrix} = \mathbf{T}_e^{-1} \mathbf{e} \quad (4)$$

where s_i and c_i are direction sines and cosines of the element edges in the undeformed state. The Cartesian strains are then related to Cartesian stresses via the constitutive equation:

$$\sigma = \mathbf{C} \mathbf{e}. \quad (5)$$

Variational analysis with strain energy (details not shown here for brevity) then leads to

$$\delta U = \frac{1}{2} (\delta \mathbf{e})^T \mathbf{C} \mathbf{e} = \frac{1}{2} (\delta \epsilon)^T \mathbf{C}_n \epsilon \quad (6)$$

where

$$\mathbf{C}_n = \frac{1}{2} \mathbf{T}_e^T \mathbf{C} \mathbf{T}_e. \quad (7)$$

Substitution of (3) into (6) and adding in the contribution of work due to external forces leads to

$$\delta \Pi = \delta U - \delta W = \frac{1}{2} (\delta \mathbf{q})^T \mathbf{B}_q^T \mathbf{C}_n \mathbf{B}_q \mathbf{q} - (\delta \mathbf{q})^T \mathbf{f}_q \quad (8)$$

from which, after integration over the element volume V , one eventually finds the simple matrix relationship

$$\mathbf{K}_q \mathbf{q} = \mathbf{f}_q \quad (9)$$

where

$$\mathbf{K}_q = V \mathbf{B}_q^T \mathbf{C}_n \mathbf{B}_q \quad (10)$$

is the element stiffness matrix and \mathbf{f}_q are covariant nodal forces along the element edges that are summed at each node. In Modelica, these equations are written as

equation

```
Kq * q = fq;
-location[1].f =
  fq[2]*Vectors.normalize(d31) -
  fq[3]*Vectors.normalize(d12);
-location[2].f =
  fq[3]*Vectors.normalize(d12) -
  fq[1]*Vectors.normalize(d23);
-location[3].f =
  fq[1]*Vectors.normalize(d23) -
  fq[2]*Vectors.normalize(d31);
```

The primary advantage of this element formulation is that because the primary variables \mathbf{q} are formed as relative displacements, the stiffness matrix \mathbf{K}_q is constant under any translation or rotation (i.e., during the entire simulation). This makes the Modelica implementation particularly simple and efficient because the stiffness matrix is computed as a parameter while relative displacements \mathbf{q} are readily calculated from the *Location* connectors of the element. The simplicity of the approach is particularly apparent in light of the complexity that would be necessary for implementing a total Lagrangian, corotational, or other geometrically nonlinear finite element formulation in Modelica.

Quadrilateral elements are formulated by overlaying two pairs of triangle elements such that the common edge of a pair crosses that of the other pair. In this case, each element has half the prescribed thickness. This is a not uncommon technique for creating quadrilateral membrane elements. Further details about the methodology used to develop the finite elements used in the *Cloth* block are discussed in a forthcoming paper (Rupp, 2018).

Discretization of the *Cloth* block is performed in Modelica by defining the locations of points on a quadrilateral patch and the number of divisions along two adjacent edges of the patch. A Coons patch (a simple bilinear interpolation approach) is then used to define the individual elements. Stiffness matrices of each element are then computed for the undeformed (stress-free) state of each element in the patch as part of the Modelica parameter evaluation procedure. A similar procedure is used to set the mass value of each lumped mass attached to the nodes, which replaces the traditional mass matrix for the element. The conversion of the mass matrix into discrete lumped masses is performed so that inertial calculations can be easily performed in the inertial (i.e., world) frame as opposed to the local frame used to perform elemental stiffness calculations. Initialization of the *Cloth* block is performed by defining parameterized initial locations of the four corners of the quadrilateral patch. In this way, the cloth can be pre-tensioned during model initialization. This mechanism also allows a folded initial blanket state to be defined, which is a common situation for many space-based solar array blankets.

The *Cloth* block is highly parameterized to allow for changes in material properties (defined via the *Properties.ClothProperty* record), thickness, undeformed shape, and folding configuration. Allowing for this design flexibility and setting up the corresponding initialization of the problem is the most complicated aspect of the *Cloth* block, as changing any one of them will affect the entire cloth formulation. To facilitate this process, several initializer functions have been created that set the cloth stiffness and mass

matrices as well as interface positions in space based on various input configuration parameters. The initializer functions can handle several different cloth configurations such as z-folds, no folds, triangle and quadrilateral discretizations, and different boundary conditions.

To improve the performance and relieve a bit of processing effort from the Modelica compiler, the element mass and stiffness matrix generation, as well as the initial placement of nodes for the *Cloth* block, is performed via a library of external “C” functions. Modelica interfaces to these functions are found in the *Initializers* subpackage.

The *Cloth* package provides a powerful modeling capability for many structures that would otherwise be difficult to model. The block has been validated through comparison to test data gathered during deployment testing of a state-of-the-art solar array (Rupp *et al*, 2016) as well as via several numerical and analytical studies.

3 Examples

We now present three examples of deployable structures that make use of the Deployable Structures Library. Each example uses the *Cloth* modeling block—meaning that they cannot be modeled using multibody dynamics simulation codes (non-finite-element-analysis based) other than Modelica or using only the MSL. Further examples can be found in Rupp *et al* (2016).

3.1 Linear Deployment Solar Array

Space-based solar arrays are ubiquitous deployable structures used in the aerospace industry as the primary source of power for satellites, space stations, space-based telescopes, etc. A common style of solar array design is the linear deployment array in which a central boom or mast is used to tension a solar blanket between two cross-bars. An example is the solar array design used onboard the ISS. These arrays are known for their compact stowage volume and high specific power.

The central mast used in linear solar arrays is the primary deployment mechanism for the array and can take many forms. The ISS arrays use a foldable truss design (Knight *et al*, 2012), the compact telescoping array concept uses a telescoping boom consisting of concentric tubes that are simultaneously moved relative to each other via a motorized mechanism (Mikulas *et al*, 2015), and the recently developed MegaROSA design uses roll-out booms (Hoang *et al*, 2016). To characterize these different types of arrays without the need to consider specific deployment mechanisms, NASA developed the Government Reference Array (GRA) as a reference design for studying the scalability of these and other high-specific-power solar array designs (Pappa *et al*, 2013).

To study this type of design, a Modelica model of the GRA was created in which the central mast was modeled using the DSL *VariableLengthBeam* block.

The cross-bars at the base and tip to which the solar blanket attaches were modeled with *FlexBeam* blocks. Each solar blanket was modeled with a *Cloth* block. The *Cloth* blocks were connected to the array structure at the trunk and tip beams only; there is no connection to the mast beam. Deployment was simulated by changing the length of the mast at a rate of 0.04 m/s. A visualization of the simulated deployment process is shown in Figure 5, and the in-plane cut force at the mast base is shown in Figure 6, where the sudden tensioning of the blanket introduces high-frequency loading. Due to the changing length of the central mast, it is not possible to perform this simulation in any other available software tool.

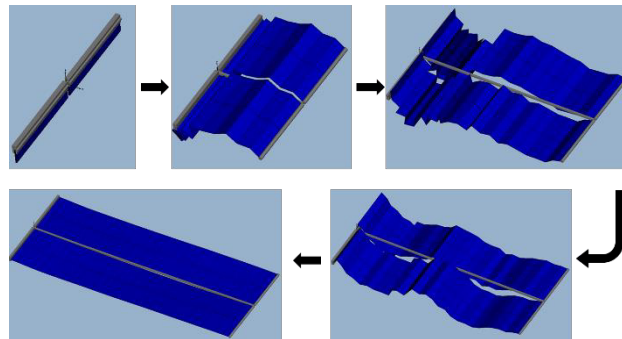


Figure 5. Deployment sequence of GRA linear solar array example.

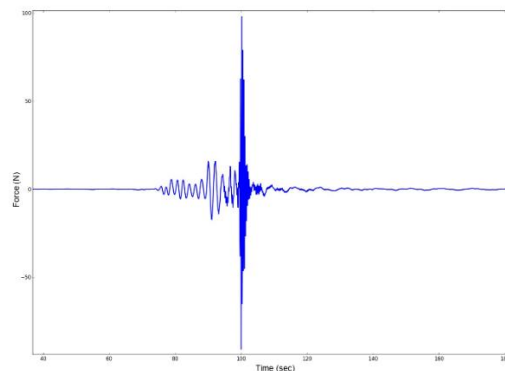


Figure 6. Transverse force at base of GRA mast during blanket tensioning event.

3.2 Origami Solar Array

One concept for large, deployable solar arrays that NASA has investigated is based on origami—the Japanese art of paper folding (Zirbel *et al*, 2013). The shape and folding patterns of these origami solar arrays are defined through three origami parameters, M, H, and R, which control the number and position of the folds as well as overall dimensions. Changing one parameter changes the design of the array, which may in turn affect stowed compactness, load distribution, and overall deployability. Thus, utilizing a model that considers these variables as design parameters is key for finding an optimal origami solar array design.

To enable rapid iteration through various design options, an origami solar array model was created using

Modelica and the *Cloth* block in the DSL. The model was parameterized based on the three origami parameters. The *Cloth* block is written such that creating and parameterizing various configurations such as this, and even completely different topologies, is simple to accomplish.

The origami array model contains only the *Cloth* block; there are no structural members, so all stiffness is provided by the cloth itself. In this case, we are exploiting the formulation of the *Cloth* in a way that each segment of the origami array consists of a single membrane element. As such, each segment acts as a semi-rigid panel without bending degrees of freedom and hinged at each edge. The model is constrained at the center of the array and is deployed by forces pulling radially outward on the tips of the array. Two configurations were studied by varying the origami parameter M , which controls the number of sides the array has when stowed. Nothing else in the model was changed. The deployment sequence for $M = 6$ is shown in Figure 7, and the deployment sequence for $M = 3$ is shown in Figure 8. Incidentally, the $M = 3$ design suffers from binding due to the geometric layout of the origami faces, which is readily apparent during the dynamic simulation.

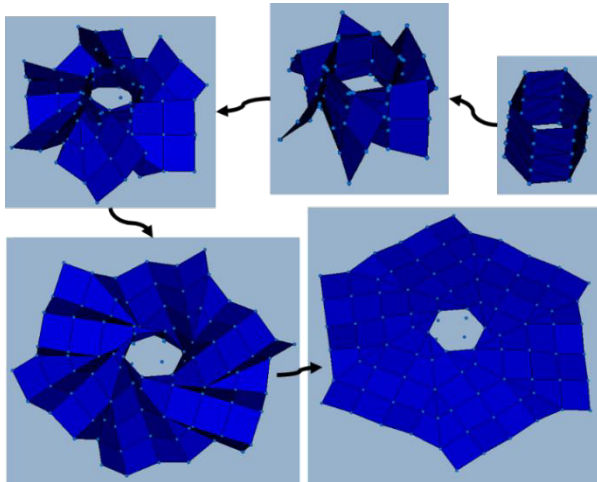


Figure 7. Origami solar array deployment sequence with origami parameters $M = 6$, $H = 2$, $R = 2$.

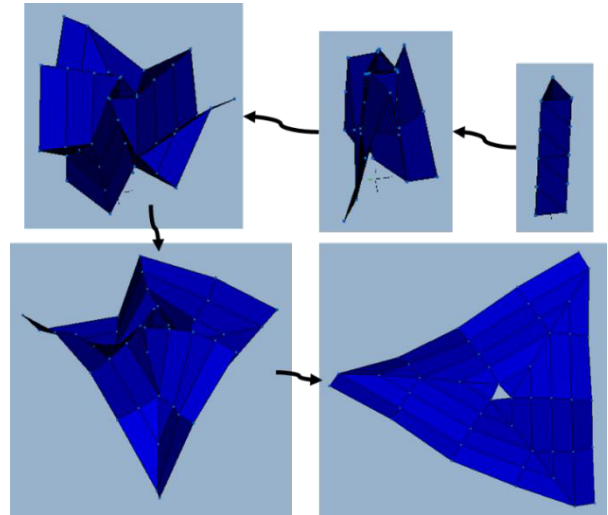


Figure 8. Origami solar array deployment sequence with origami parameters $M = 3$, $H = 2$, $R = 2$.

3.3 Solar Sail

Solar sails have long been proposed as a space propulsion technology, but only recently with the availability of inexpensive cubesat platforms have they been demonstrated in actual spaceflight. As such, solar sail design has yet to gain much flight heritage, and the means by which a sail is deployed can vary widely between design concepts. Solar sails are large, gossamer-thin reflective blankets that harness solar radiation pressure to exert small amounts of thrust on the spacecraft. Integrated over time, the thrust can translate into significant accelerations, possibly enabling interstellar travel (Landis, 1999).

One particular concern of solar sails is the blanket tensioning process. Because the sails are thin (on the order of tenths of millimeters), they are prone to tearing if excessive force is applied. However, if the sails are not taut, wrinkles in the reflective fabric will diffusely reflect solar radiation, resulting in a loss of momentum imparted by the radiation pressure.

A Modelica model using the *Cloth* block and the *VariableLengthBeam* block was created to examine forces exerted on the blanket during the tensioning process. The sail was modeled as a single, square *Cloth* block. The four deploying booms were modeled with *VariableLengthBeam* blocks fixed to each other at the center of the structure where the spacecraft bus would be located, and the sail was connected to the booms through extensional springs at each corner of the sail. This simulation started at the instant when the sail is fully deployed but not yet tensioned and ended when the booms reach their final predetermined length. The start and end states of the sail are depicted in Figure 9. The sail, which is initially square in its undeformed state, stretches at the corners and exhibits a catenary shape along its edges. This is the expected behavior for such a system and demonstrates how the DSL *Cloth* model can

be used to perform structural analysis of solar sail structures.

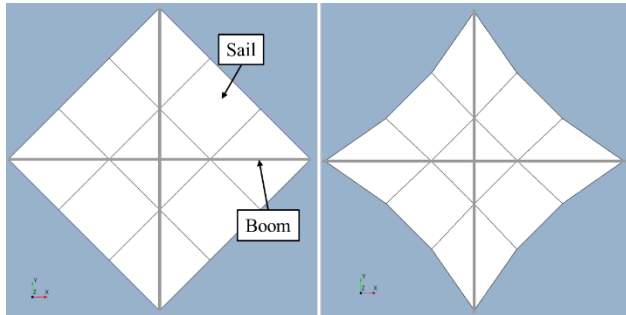


Figure 9. The solar sail model in its untensioned (left) and tensioned states (right).

4 Conclusions

This paper presents a Modelica library for the analysis of deployable structures. The library provides modeling blocks useful for creating models of deployment mechanisms and structures unique to deployable systems, particularly those used on spacecraft. In particular, a modeling capability for structural cloths or fabrics is presented that is not available in any other multibody dynamics software package.

The Deployable Structures Library is open-source and available via the github page <https://github.com/ATAEngineering/DeployStructLib>, which will likely be linked to via the Modelica Association website at <https://modelica.org/libraries>. While the library should work for any Modelica implementation per the Modelica standard, it was developed using OpenModelica and has not been tested using other software. External contributions and bug fixes or reports are encouraged.

Acknowledgements

This work was funded under Phase I and Phase II Small Business Innovation Research (SBIR) contracts sponsored by the National Aeronautics and Space Administration (NASA). The authors would like to thank technical monitors Geoffrey Rose and Richard Pappa at NASA Langley Research Center.

References

- C. Fellipa. 2017. Introduction to Finite Element Methods course material, Chapter 15: Three-Node Plane Stress Triangles. <https://www.colorado.edu/engineering/CAS/courses.d/IFEM.d/IFEM.Ch15.d/IFEM.Ch15.pdf>
- G. Ferretti, A. Leva, and B. Scaglioni. Object-oriented modelling of general flexible multibody systems. *Mathematical and Computer Modeling of Dynamical Systems*, 20(1): 1-22, 2014. doi: 10.1080/13873954.2013.807433.
- A. Heckmann, M. Otter, S. Dietz, and J.D. López. The DLR FlexibleBodies library to model large motions of beams and of flexible bodies exported from finite element programs. In *Proceedings of 5th Modelica Conference*. Vienna, Austria, September 2006.
- B. Hoang, W. White, B. Spence, S. Kiefer. Commercialization of Deployable Space Systems' roll-out solar array (ROSA) technology for Space Systems Loral (SSL) solar arrays. In *Proceedings of 2016 IEEE Aerospace Conference*. Big Sky, MT, 2016. doi: 10.1109/AERO.2016.7500723.
- N. F. Knight Jr., K. B. Elliott, J. D. Templeton, K. Song, J. T. Rayburn. FAST Mast Structural Response to Axial Loading: Modeling and Verification. In *Proceedings of 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*. Honolulu, HI, April 2012. doi: 10.2514/6.2012-1952.
- G.A. Landis. 1999. Advanced Solar- and Laser-pushed Lightsail Concepts. Final Report, NASA Institute for Advanced Concepts. http://www.niac.usra.edu/files/studies/final_report/4Landis.pdf
- M. Mikulas, R. Pappa, J. Warren, and G. Rose. Telescoping Solar Array Concept for Achieving High Packaging. In *Proceedings of the 2nd AIAA Spacecraft Structures Conference*. Kissimmee, FL, January 2015.
- R. Pappa, G. Rose, T. Mann, J. Warren, M. Mikulas, T. Kerslake, T. Kraft, J. Banik. Solar Array Structures for 300 kW-Class Spacecraft. *Space Power Workshop*, 2013. (<https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20140000360.pdf>)
- C. J. Rupp. 2018. The Relative Finite Element Method. In Prep.
- C. J. Rupp, L. Schweizer, and D. Murphy. Rapid Parametric Analysis and Design of Space-Based Solar Arrays. In *Proceedings of the 3rd AIAA Spacecraft Structures Conference*. San Diego, CA, January 2016.
- F. Schiavo, L. Viganò, and G. Ferretti. Object-oriented modelling of flexible beams. *Multibody System Dynamics*, 15(3): 263–286, 2006. doi: 10.1007/s11044-006-9012-8.
- J. Wright. 2007. <https://www.nasa.gov/content/astronaut-scott-parazynski-works-near-solar-array>
- S. Yang, Z. Deng, J. Sun, Y. Zhao, and S. Jiang. 2017. A Variable-Length Beam Element Incorporating the Effect of Spinning. *Latin American Journal of Solids and Structures*, 14(8): 1506-1528. doi:10.1590/1679-78253894
- S. A. Zirbel, R. J. Lang, R. W. Thomson, D. A. Sigel, P. E. Walkemeyer, B. P. Trease, S. P. Magleby, L. L. Howell. Accomodating Thickness in Origami-Based Deployable Arrays. *Journal of Mechanical Design*, 135, 2013. doi: 10.1115/1.4025372.

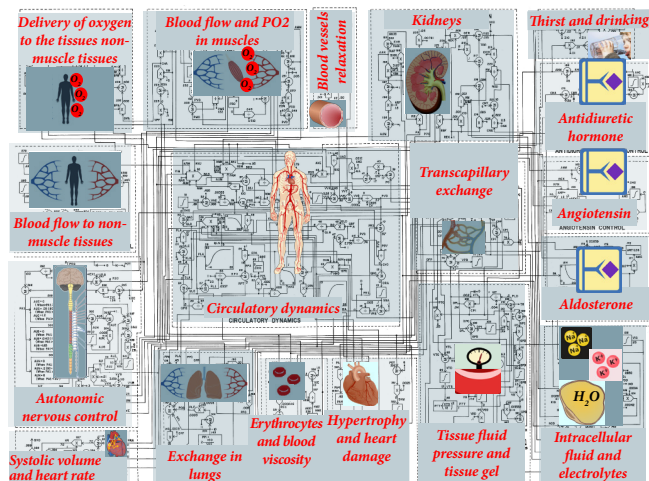


Figure 3. Interconnected physiological subsystems in the Guyton's model (Guyton et al., 1972).

A monography (Arthur C. Guyton, Jones, & Coleman, 1973) in which many of the approaches were explained in more detail was published a year later and a further monograph, by Guyton et al. (Arthur C. Guyton, Taylor, & Granger, 1975), presenting a reasonably detailed explanation of the mathematical formalization of the body fluid dynamics description, appeared in 1975.

2. Formalization of physiological relations – the PHYSIOME Project

Guyton's model was a milestone of sorts, applying a system approach to physiological regulations and describing the dynamics of interrelations between/among physiological subsystems by means of a system of graphically represented mathematical equations. Guyton's graphical diagram marked the emergence of an area of physiological research into the interconnected physiological systems in the living body, now referred to as “integrative physiology” (Coleman & Summers, 1997; Mangourova, Ringwood, & Van Vliet, 2011; Reinhardt & Seeliger, 2000).

Much as how theoretical physics strives to describe physical reality and explain the results of experimental research, “integrative physiology” tries, based on experimental results, to set up a formalized description of the interconnections of physiological regulations and explain their function both in the healthy body and during the development of various diseases.

Formalization, i.e. replacement of a verbal description of physiological systems with the precise language of mathematics, is closely linked to the issue of computer modeling. It is an asset of the formal description that deductions regarding the behavior of a system described by formalized tools are made based on the rules of a formalized language, i.e. by solving the equations of a mathematical model. This is a task that can be left to a computer – the computer solves equations describing the biological reality – and so it is computer simulation that is involved.

The concept of formalization started later and progress is somewhat slower in biological and medical sciences than in physics, chemistry and technology, because biological systems are much more complex. While the formalization process in physics started as early as the 17th century, formalization in medical and biological sciences came only together with cybernetics and computer science. This field of science uses computer models set up based on a mathematical description of the biological reality.

Formalized description of physiological systems is currently the subject of the international **PHYSIOME** Project (<http://www.physiome.org>), successor to the **GENOME** Project whose outcome consisted in a detailed description of the human genome. The aim of the PHYSIOME Project is to provide a formalized description of physiological functions (Bassingthwaite, 2000; P. Hunter, 2016; Peter J. Hunter, Crampin, & Nielsen, 2008; P. J. Hunter, Li, McCulloch, & Noble, 2006; P. Hunter, Robbins, & Noble, 2002; Omholt & Hunter, 2016). Physiome makes efforts to apply the formalized approach in order to integrate our knowledge, from the cell level to the organ level to the whole-body level, with a view to gaining insight into how all that works as a whole. The European initiative in this area is represented by the The International Union of Physiological Sciences (IUPS) <http://www.iups.org/physiome-project/>. The work of the IUPS Physiome Project has been boosted by the European Commission-funded **VIRTUAL PHYSIOLOGICAL HUMAN INITIATIVE** project (under the virtual physiological human institute <http://www.vph-institute.org/>), aiming, among other things, to apply the formalized approach to human physiology in clinical medicine and to use computer models in pre-clinical trials.

Integrative models of laboratory animals have been developing lately in addition to the integrative models of human physiology. For example, the aim of the **VIRTUAL RAT** project (<http://www.virtualrat.org/>) is to set up a complex model of the laboratory rat, which can readily be validated against experimental data on laboratory animals (Beard et al., 2012).

3. New modeling environments

In the meantime, general software simulation environments emerged, enabling models to be developed in a graphical format and allowing them to be debugged and ultimately verified. Among them is the widely used Matlab/Simulink tool from Mathworks, enabling a simulation model to be composed from various pre-defined components visually by drag-and-drop into the simulation networks. The Simulink blocks are very similar to the elements used by Guyton for a formalized representation of physiological relations. They actually differ in the graphical format only.

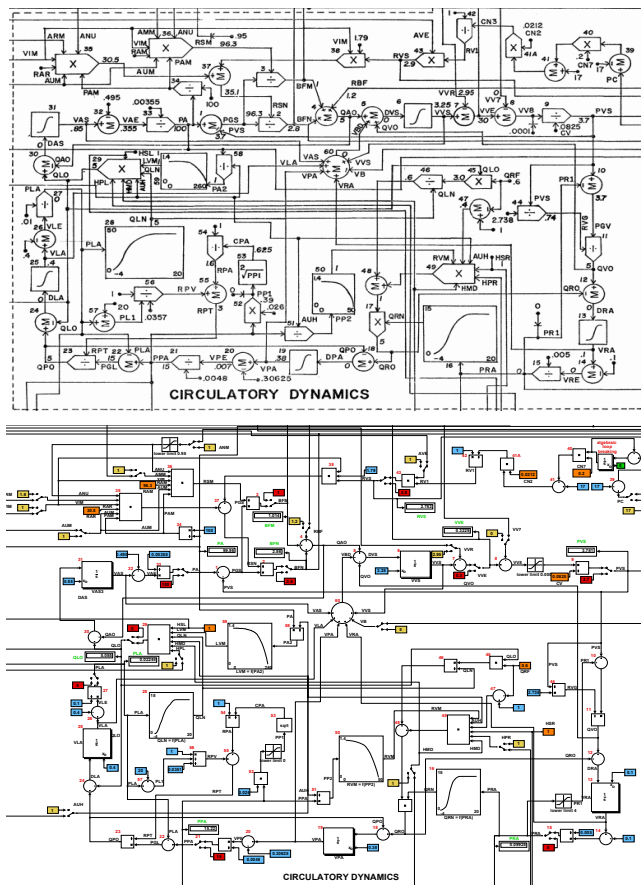


Figure 4. Circulatory dynamics - detailed representation of the central structure of the Guyton model in the original graphical notation (upper part of the figure) and in our Simulink implementation (bottom of the figure), which shows blood flows through aggregated parts of the circulatory system, and action of the heart as a pump (Jiří Kofranek & Rusz, 2010).

This similarity of Guyton’s approach and philosophy of Simulink software inspired us to revive the old traditional Guyton’s diagram by means of Simulink and transform it into a functional simulation model (Jiří Kofranek & Rusz, 2010). The appearance of the Simulink model could be nearly identical with that of the initial layout (Fig. 4).

Simulation visualization contained a number of errors (or rather “graphical typos”) in the initial layout. This poses no problem in a drawn picture, but the moment you try to revive it in Simulink, the model collapses as a whole immediately.

4. Model presentation in scientific publications

Guyton’s diagram is just an illustrative picture condensing a system of equations describing a complex model into a graphic form. Since the description contained errors, it was difficult to reproduce the model based on the graphical diagram only. However, the authors made the model program in Fortran available on request, which en-

abled the behavior of the model to be tested.

This however introduces additional problem in versioning - the published version may produce different results than the later, usually further updated, provided on request.

It is a generally adopted principle that if a result is described in a scientific journal, then the experimental design must be reproducible at another workplace. The reproducibility principle plays a key role in scientists’ efforts to disclose the secrets of Nature.

Actually, however, it is often violated in scientific publications dealing with biomedical models. This is not always a mistake of the authors – frequently just some letter or index is omitted, and it is then very difficult for the reader to understand the model or even to implement it.

The reviewers do not reimplement the models from the description (as they usually have the underlying code accessible on request) and thus the equations could easily contain a mistake.

Also, biomedical models are often so complex that the limited space allocated for the paper allows the authors just to present the basic model equations (and sometimes not all of them) and no space remains for additional information (starting values of variables of state, all parameter values, solver settings etc.) that is needed to set up the model at another workplace. Also, a number of articles do synthesise multiple models together, be it an extension of their previous research or adopted from literature. The details of combining the old (and referenced) with the newly presented do often raise a number of issues.

From our teaching experience, around 80 % of models implemented based solely on a description in a published article were incomplete or contained some error, which makes the model unusable. Nielsen et al. also support our observation of difficult reproducibility (Nielsen, Nilsson, & Matheson, 2012). A scientific paper describing a model should thus be accompanied by a digital enclosure (accessible on the Internet) containing a detailed description of the model structure, including the values of all parameters and most of all containing a complete source code in a common, formal programming language, adequate for the reader to be able to run the model, reproduce the model results and to potentially use the model as a basis for their own work where appropriate. The sharing of the complete source code is becoming common practice and even a requirement in a number of journals publishing scientific papers on computer models, especially the open-access ones.

5. Repositories of biomedical system models

A serious obstacle arises if a model is published in a

modeling language requiring a commercial license (such as Matlab/Simulink by MathWorks), because the reader must be a licensee of the particular system to be able to just reproduce the model results.

This is why considerable efforts have been made within the international **PHYSIOME** Project to create simulation languages appropriate for describing biomedical models and saving them in specific databases – model repositories. Publicly available tools for creating and launching models programmed in such languages were also created in this context.

So, for instance, the **Virtual Cell** project (<http://vcell.org>) has been set up for visualization and simulation of the cell metabolism and cell signal paths. That project was developed by the Center for Cell Analysis & Modeling, at UConn Health, University of Connecticut (USA). Quite a large group of users exists now around that project. The Virtual Cell developmental environment is interlinked with a number of databases and with the list of diverse models. This environment works on the client-server principle.

The **“Bio Tapestry”** project of Caltech (California Institute of Technology), Eric Davidson’s laboratory, is designed for modeling regulatory gene networks (where expression of the various genes is blocked/activated by transcription factors which, in turn, result from the expression of other genes) (<http://www.biotapestry.org>). Regulatory gene networks look sort of like status automata (gene expression depending on the presence of the relevant transcription factors) – the gene expression may result in the formation of a protein, which can also be a transcription factor. A gene network editor and simulator can visualize the stepwise changes in the expression of the various genes and, based on a comparison with the experimental data, help explain the complex processes taking place particularly during embryonic development. Once again, this tool is interlinked with electronic model archives and has its own user community.

Two large global centers maintaining extensive physiological model databases are currently involved in the **PHYSIOME** Project.

The first center (founded by Jim Bassingthwaight) is administered by Washington University in Seattle and uses the specifically created **JSim language** for the model database (Butterworth, Jardine, Raymond, Neal, & Bassingthwaight, 2013). A description of the language, installation sources and tutorials are available at: <http://www.physiome.org/jsim>.

The environment for the creation and launching of models written in JSim is based on Java, owing to which it can be easily installed on different platforms. This environment can be used to modify and launch models from an extensive model database: <http://www.physiome.org/>

[jsim/models](http://www.physiome.org/jsim/models).

The other large physiological system model database is maintained by the University of Auckland, New Zealand (<https://unidirectory.auckland.ac.nz/profile/phun025>). Petr Hunter, founder of the database, has built a top-ranking workplace in New Zealand – halfway between America and Europe (<http://www.abi.auckland.ac.nz/en.html>).

This institution uses the **CellML language** (Cooling & Hunter, 2015; Cuellar et al., 2003; Garny et al., 2008; Lloyd, Lawson, Hunter, & Nielsen, 2008) to describe the models: <http://www.cellml.org>. The tools for browsing, creating and launching models in CellML are available at <https://www.cellml.org/tools>. A tool for converting from CellML to JSim also exists. OpenCell is a tool for CellML simulation: <https://www.cellml.org/tools/opencell>. A large database of models has been created in CellML and is available at: <https://models.physiomeproject.org>. The models were taken from the literature and reprogrammed into CellML (or JSim). Each model is accompanied by reasonably detailed documentation. A model downloaded from the database in CellML can be simulated in the OpenCell environment.

However, the development of specialized simulation tools is limited by the funding allocated for the physiological research.

6. Equation-based languages

Both JSim and CellML are causal, block-oriented languages. The same characteristics also applies to Simulink (from Mathworks), frequently used to model biomedical systems.

The main problem with block-oriented languages lies in the fact that a simulation network consisting of hierarchically connected blocks is a graphical representation of a chain of input value transformations to output values. This means that an exact algorithm for the calculation chain from the input values to the output values must be defined when creating a model.

As a consequence of the requirement of a fixed connection direction from the inputs to the outputs, **the connection of the blocks reflects the calculation procedure rather than the structure of the modeled reality itself.**

Where complex models are involved, deriving the causality of the calculation (i.e. deriving the algorithm for calculation of the output variables from the input variables) is by no means a straightforward task.

This problem is addressed by modern equation-based, or acausal, modeling languages. Unlike block-oriented languages, where the structure of the hierarchic block connections represents more the calculation method than the reality being modeled, the structure of the models in Modelica reflects the very structure of the reality mod-

7. Modelica – a language suitable for publishing and sharing biomedical models

Modelica, initially developed as an academic project in collaboration with small developmental companies at the universities in Lund and in Linköping, soon emerged as a highly effective and efficient tool for modeling complex models with potential application in mechanical engineering and in the automotive and aircraft industries.

Owing to this, the development of Modelica eventually gained support from the commercial sector, but the language itself is developed by an independent nonprofit association (see www.modelica.org). The Modelica Association gathers a number of key commercial as well as academic players, which ensures the stability of the platform and its relative independence on business decision of individual companies.

The speed at which this new simulation language spread to the various industries and was adopted by diverse commercial developmental environments is striking. Thanks to adoption by commercial sector, the language and both proprietary and open-source tools are already mature enough to guarantee reliable modeling platform. Several commercial as well as noncommercial developmental tools using this language currently exist (see www.modelica.org/tools).

Modelica users are therefore not confined to licensed commercial developmental tools: in fact, mature open-source developmental tools for this language exist now (e.g. OpenModelica, available at openmodelica.org, and JModelica, available at jmodelica.org/).

Thus, the effort spent on developing and maintaining own simulation platform is now unnecessary. Development driven by a number of high-tech industry (automotive, energy and aerospace) also guarantees small risk of stale development or platform discontinuation.

Modelica therefore appears to be a highly promising tool for publishing and sharing models. Some researchers have already adopted Modelica as their preferred modeling tool, e. g. (Heinke, Pereira, Leonhardt, & Walter, 2015; Maksuti, Bjällmark, & Broomé, 2015), even switched from the SimScape (de Canete, 2015), or use the general model exchange functional mockup interface, based on Modelica initiative (Gesenhues et al., 2017). The developers of the most complete physiological model Hummod are also considering using Modelica implementation to make the model easier to maintain (R. Hester, personal communication, August 2018). However, the penetration of Modelica in physiological research is still not massive.

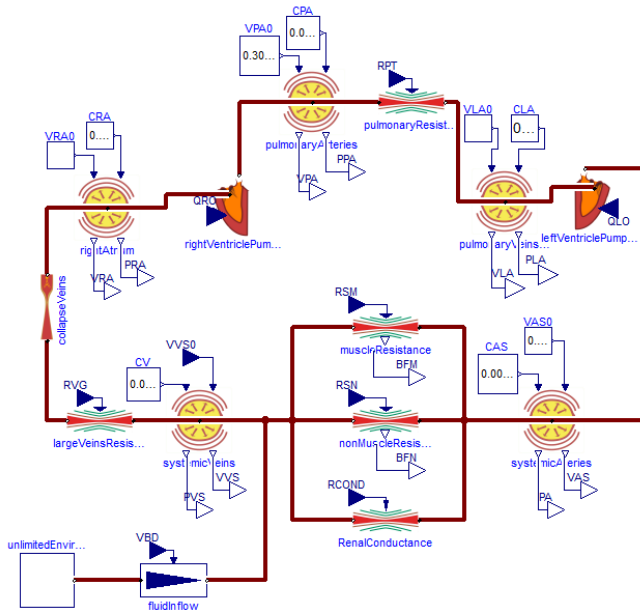


Figure 5. The same part of the model as in Figure 4, but implemented in Modelica. The model contains connected instances of two pumps (of the right and left heart ventricle), elastic vascular compartments, and resistances. Upon its comparison with Fig. 3, it can be seen that the model structure in Simulink corresponds rather to a computational algorithm, while the model structure in Modelica shows more of the structure itself of the modelled reality. Figure was adapted from (Kofránek, Mateják, & Privitzer, 2011).

eled (compare Fig. 4 in block-oriented language and Fig. 5 in Modelica). Owing to this, even complex models are adequately transparent and understandable in Modelica (Ježek, Kulhánek, Kalecký, & Kofránek, 2017).

A model should be *understandable* not only to the development team members but also to others. If only the authors understand their model, they will hardly obtain the necessary feedback or new impulses for their work from the scientific community.

This is of great importance with respect to the creation of complex integrative physiology models. When using block-oriented languages (be it Simulink or specifically created open-source languages for the documentation of biomedical models – JSim or CellML), the resulting complex program is not very comprehensible. It is largely only the authors who are able to understand their complex models. Modelica solves this problem efficiently, and integrative models of human physiology in Modelica have the potential for wider use within the scientific community.

Nowadays, the principles of equation-based approach are further implemented also in other products. E.g. SimScape (Mathworks, MA, USA) software package extends the commonly used Matlab/Simulink environment with the multidomain physical system modeling capabilities, useful also for biomedical engineering (de Canete, Saz-







| Connector: | | flow variable | nonflow variable |
|---|------------|---|---|
|  | Chemical | molar flow [mol.s ⁻¹] | concentration [mol.m ⁻³] |
|  | Hydraulic | volumetric flow [m ³ .s ⁻¹] | pressure [Pa] |
|  | Thermal | heat flow [W] | temperature [K] |
|  | Osmotic | volumetric flow [m ³ .s ⁻¹] | osmolarity [mol.m ⁻³] |
|  | Population | change [s ⁻¹] | size [1] |
|  | Electrical | electric current [A] | electric potential [V] |

Table 1. Physical connectors in Physiobrary compared with electrical connector in the Modelica Standard Library. Each connector in Physiobrary defines one physical domain. As seen in Table 2, most of the components have analogies throughout the domains. For example, the resistor in electrical circuits has an analogy in the chemical domain as diffusion, because the molar flow of a substance is driven by the concentration gradient in the same way an electric current is driven by the voltage gradient.

8. Application libraries for biomedical simulations in Modelica

The proliferation of Modelica was facilitated by the existence of libraries for the most diverse areas, which appreciably simplify model formation for the given application domain. A model is set up by interconnecting instances of library components, like – figuratively speaking – buildings made of Lego bricks.

The majority of current libraries serve physical and technological applications. New libraries had to be created for models in the biomedical domain.

This is why we have created *Physiobrary* (Marek Matejak et al., 2014), intended for model creation in physiology (see <http://physiobrary.org>).

Physiology is a very progressive discipline, that examines how the living body works. And there is no surprise, that all processes in the human body are driven by physical laws of nature in several physical domain (see Tab 1 and Tab 2). And it is a great challenge to join many old empirical experiments with the ‘new’ physical principles. We hope, that this library helps the unflagging effort of Physiologists to exactly describe the processes and include their hypothesis.

The Physiobrary contain basic physical laws in Human Physiology usable for cardiovascular circulation, metabolic processes, nutrients distributions, thermoregulation, gases transport, electrolytes and acid-base regulations, water distributions, hormonal or pharmacological regulations.

Chemical processes also have to be modeled in the bio-









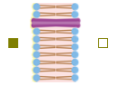





| Resistance | Accumulation | Stream |
|--|---|--|
| $f_i = G^*(e_1 - e_2)$ $f_1 + f_2 = 0$ | $\int f = a$ $a = C^*e$ | $f_1 = \begin{cases} F e_1, & F \geq 0 \\ F e_2, & F < 0 \end{cases}$ $f_1 + f_2 = 0$ |
| G..conductance | C..capacitance | F..stream flow |
|  Chemical diffusion |  Substance |  Solution flow |
|  Hydraulic resistance |  Elastic vessel | not applicable |
|  Heat convection |  Heat |  Heated mass flow |
|  Semipermeable membrane |  Osmotic cell | not applicable |
| not applicable |  Population |  Growth, Differentiation |
|  Electrical resistor |  Electrical capacitor | not applicable |

Table 2. Analogies of selected Physiobrary components based on connectors from Table 1 compared with electrical components in the Modelica Standard Library. To define the mathematical analogies in Table 2 we use the symbols e for effort (for connector nonflow variables) and f for flow (for connector flow variables). If there are more connectors in a component, they are differentiated by index. Unfortunately many elementary components in Physiobrary do not have analogies through these domains. The special definitions in Physiobrary include, for example, the components for chemical reaction, for hydrostatic pressure, for Henry’s solubility of gas in liquid, for Donnan’s equilibrium of electrolytes on membrane etc. Table 1 and 2 were presented in (M. Matejak & Kofranek, 2015).

medical area, and so we created the *Chemical* library as well (Matejak, Tribula, Jezek, & Kofranek, 2015).

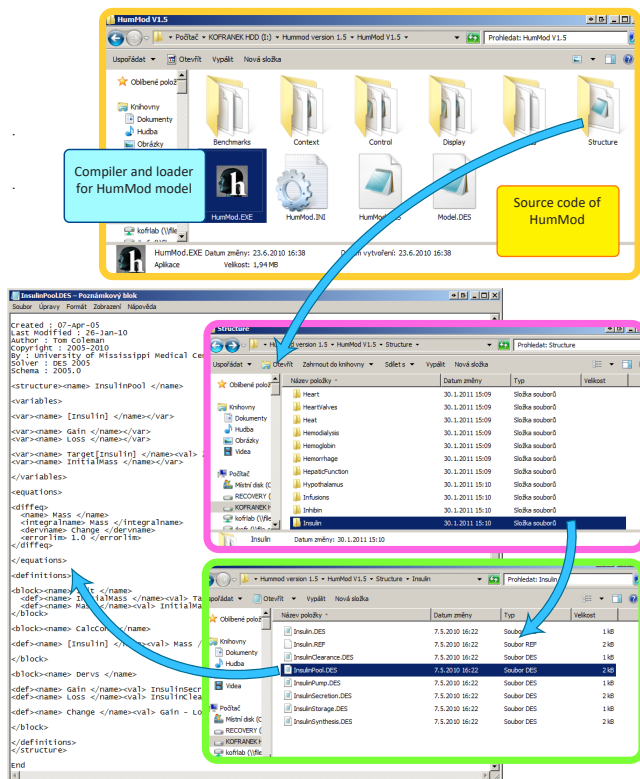


Figure 6. HumMod simulator has been distributed with a compiler, loader and the source code written in thousands of XML files.

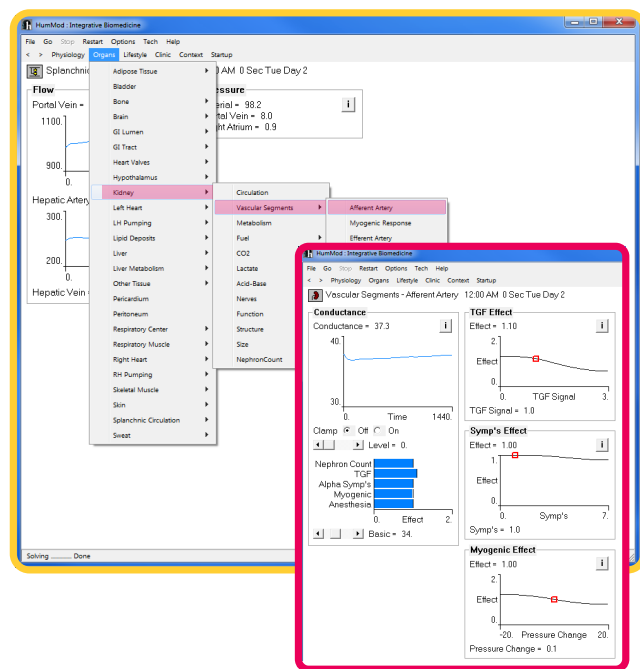


Figure 7. The user can compile and run the HumMod model. Using a widely branched menu, hundreds of variables can be monitored during simulation experiments.

Those libraries are a result of our many years' experience in the implementation of extensive hierarchical models of human physiology in Modelica (Ježek et al., 2017; Jiri Kofranek, Matejak, & Privitzer, 2011), the HumMod model in particular.

The *HumMod* model, set up in international collaboration by a group of collaborators and disciples of A. Guyton at the Mississippi University Medical Center, USA, (R. Hester, Brown, Husband, & Iliescu, 2011; R. L. Hester, Coleman, & Summers, 2008) is probably the most extensive existing model of integrated physiological systems of human physiology. The authors do not keep the structure secret: the model source text (containing over 5,000 variables) can be downloaded from the project web pages: <http://hummod.org>. The source text is written in a specific XML markup language. The whole mathematical model is offered as an open-source tool. The user is free to download both the source text and the translator into their computer from the web page and to launch the model on their own computer (Fig. 6 and 7). The user is in a position to modify the model to suit their purpose. A problem is in the fact that the XML source texts of the entire model are written in thousands of files located in hundreds of folders, and gaining insight into the mathematical relation by browsing through thousands of interlinked XML files is very difficult.

It appears that the *comprehensibility of the descriptions of complex integrative models* is one of the factors limiting their adoption by the scientific community. If the creators are the only ones to understand their model, any possibility of technical communication with other scientists is considerably limited. And so is the potential for a wider use within the broad scientific community. So, the development of methodologies that will make the description of the structure of complex hierarchical models so clear that a wide group of users can understand it is gaining in importance.

Specific browsers allowing the relations in the model to be browsed have been created in order to facilitate understanding of the HumMod model (Wu, Chen, Pruetz, & Hester, 2013). Even so, the equations in the model and their interrelations are rather difficult for the user to understand. One of the ways to make understanding complex hierarchic models easier is to use the Modelica language. This is why we decided to re-implement the entire complex model of the US authors in Modelica.

Model re-implementation in Modelica makes the model structure much clearer (see Fig. 8), the source code resembling hierarchic physiological schemes. Making the model clearer also helped detect some errors in the initial US implementation of the HumMod model. We modified HumMod and extended it mainly in the area of modeling blood gas transfer and homeostasis of the inner environment, the acid-base equilibrium in particular (Jiri Kofranek et al., 2011).

Our version of the HumMod model, called *Physi-omodel*, is being developed as an open-source tool. The *model source texts* (i.e. equations, values of all constants, etc.), which constitute a formalized representation of the

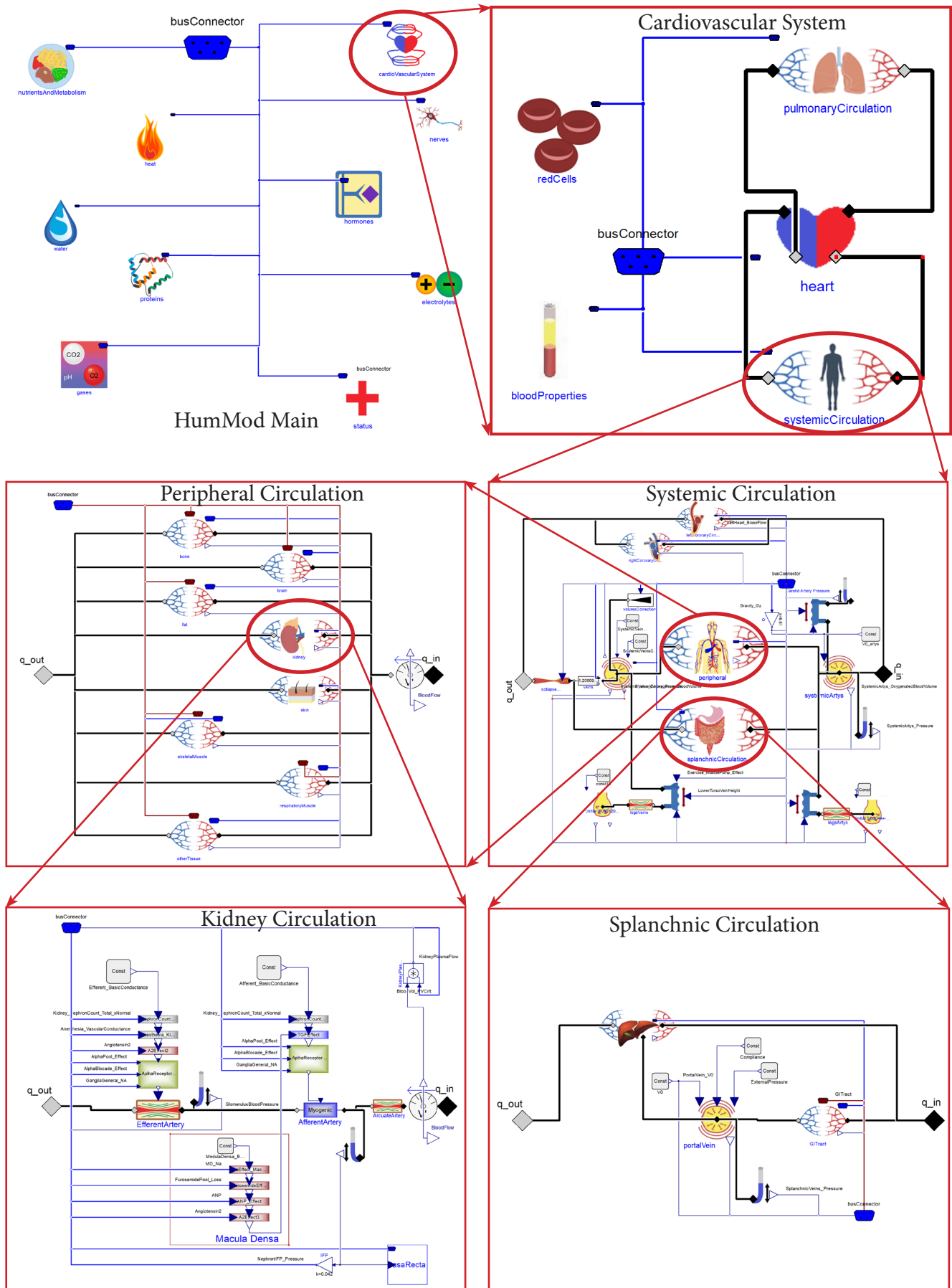


Figure 8. Illustration of a part of the source text of our HumMod implementation in Modelica. The source text resembles hierarchical physiological schemes. Image adapted from <http://www.physiomodel.org/>

physiological relations, are *publicly available* at <http://www.physiomodel.org>.

9. Community for biomedical model evolution in Modelica

International cooperation in combination with the openness of result sharing is a driving force of scientific development in today's globalized world. As experience shows, the development of complex software systems is conditional on the existence of the widest possible user/developer community to provide feedback and ensure further innovation of the complex products through cooperative efforts, with the ultimate use in commercial applications. This is why the development of projects with open source codes has become so popular lately.

The creation of extensive models of physiological systems is not just a purely scientific issue: they also potentially can be used in commercial applications. Medical simulators, which are based on a validated human physiology model (like aircraft simulators are based on an adequately faithful aircraft model), are a good example.

The development of complex integrated physiology models may be optimal if open scientific development is combined with the exploitation of business opportunities and financing by the commercial sector.

The creation of *OpenModelica* (<https://openmodelica.org>) within an open community may serve as a model in this area. The development of the products is managed by a consortium joining together universities, commercial companies and individual developers (*Open Source Modelica Consortium* - see <https://openmodelica.org/home/consortium>). The consortium members include both large companies and small developmental enterprises. Research is funded from member fees, the height of which depend on the company size as well as on the number of sold products developed by using the OpenModelica licenses. A reasonably large community combining users with many cooperating developers has concentrated around OpenModelica, resulting in a well-performing open-source product which is competitive in the existing environment of expensive commercial Modelica implementations. Commercial companies are free to use any part of the OpenModelica environment and expand it as appropriate, also during the development of competitive commercial implementations of the Modelica language (this is why companies such as Wolfram and MapleSoft have joined the consortium).

It is conceivable that an association of the academic community and commercial companies built on similar foundations and called, say, "*Physiomodelica Open Source Consortium*", may ensure further development of integrative physiology models in the future.

8 Conclusions

Let us sum up the factors owing to which Modelica is a language suitable for publishing and sharing biomedical models:

1. Modelica is a modeling language, not a proprietary product owned by a commercial company (such as, e.g. Mathworks' Matlab and Simulink).
2. Publicly accessible noncommercial developmental tools (such as OpenModelica and JModelica) exist for Modelica and are mature and reliable enough, the development is driven by well funded industries.
3. Modelica includes application libraries facilitating biomedical system modeling.
4. The model structure in the acausal Modelica language is clear, reflecting more the structure of the original modeled than that of the calculation and enabling extensive hierarchic models to be set up.
5. Modelica may be broadly used in a number of application domains. Further Modelica developments are aimed at satisfying the requirements of the industries and are not dependent on grant funds from the PHYSIOME Project.

Acknowledgements

The authors appreciate the partial funding of this work by PRVOUK P/24/LF1 and MPO FV20628

References

- Bassingthwaighe, J. B. (2000). Strategies for the physiome project. *Annals of Biomedical Engineering*, 28(8), 1043–1058.
- Beard, D. A., Neal, M. L., Tabesh-Saleki, N., Thompson, C. T., Bassingthwaighe, J. B., Shimoyama, M., & Carlson, B. E. (2012). Multiscale modeling and data integration in the virtual physiological rat project. *Annals of Biomedical Engineering*, 40(11), 2365–2378.
- Butterworth, E., Jardine, B. E., Raymond, G. M., Neal, M. L., & Bassingthwaighe, J. B. (2013). JSim, an open-source modeling system for data analysis. *F1000Research*, 2, 288.
- Coleman, T. G., & Summers, R. L. (1997). Using mathematical models to better understand integrative physiology. *Journal of Physiology and Biochemistry*, 53, 45–46.
- Cooling, M. T., & Hunter, P. (2015). The CellML Metadata Framework 2.0 Specification. *Journal of Integrative Bioinformatics*, 12(2), 260.
- Cuellar, A. A., Lloyd, C. M., Nielsen, P. F., Bullivant, D. P., Nickerson, D. P., & Hunter, P. J. (2003). An Overview of CellML 1.1, a Biological Model Description Language. *Simulation*, 79(12), 740–747.
- de Canete, J. F. (2015). Object-Oriented Programming for Modeling and Simulation of Systems in *Physiology*. *International Journal of Medical, Health, Biomedical, Bioengineering and Pharmaceutical Engineering*, 9(4),

- 343–346.
- de Canete, J. F., Saz-Orozco, P. del, Moreno-Boza, D., & Duran-Venegas, E. (2013). Object-oriented modeling and simulation of the closed loop cardiovascular system by using SIMSCAPE. *Computers in Biology and Medicine*, 43(4), 323–333.
- Garny, A., Nickerson, D. P., Cooper, J., Weber dos Santos, R., Miller, A. K., McKeever, S., ... Hunter, P. J. (2008). CellML and associated tools and techniques. *Philosophical Transactions. Series A, Mathematical, Physical, and Engineering Sciences*, 366(1878), 3017–3043.
- Gesenhues, J., Hein, M., Ketelhut, M., Habigt, M., Rüschen, D., Mechelinck, M., ... Abel, D. (2017). Benefits of object-oriented models and ModeliChart: modern tools and methods for the interdisciplinary research on smart biomedical technology. *Biomedizinische Technik. Biomedical Engineering*, 62(2), 111–121.
- Guyton, A. C., Coleman, T. G., & Granger, H. J. (1972). Circulation: overall regulation. *Annual Review of Physiology*, 34, 13–46.
- Guyton, A. C., Jones, C. E., & Coleman, T. G. (1973). *Circulatory physiology: cardiac output and its regulation*. Philadelphia: WB Saunders, 1973.
- Guyton, A. C., Taylor, A. E., & Granger, H. J. (1975). *Circulatory Physiology II. Dynamics and control of the body fluids (Vol. 2)*. Saunders.
- Heinke, S., Pereira, C., Leonhardt, S., & Walter, M. (2015). Modeling a healthy and a person with heart failure conditions using the object-oriented modeling environment Dymola. *Medical & Biological Engineering & Computing*, 53(10), 1049–1068.
- Hester, R., Brown, A., Husband, L., & Iliescu, R. (2011). HumMod: a modeling environment for the simulation of integrative human physiology. *Frontiers in Physiology*. Retrieved from <http://journal.frontiersin.org/article/10.3389/fphys.2011.00012>
- Hester, R. L., Coleman, T., & Summers, R. (2008). A multilevel open source integrative model of human physiology. *The FASEB Journal*, 22(1 Supplement), 756.8–756.8.
- Hunter, P. (2016). The Virtual Physiological Human: The Physiome Project Aims to Develop Reproducible, Multiscale Models for Clinical Practice. *IEEE Pulse*, 7(4), 36–42.
- Hunter, P. J., Crampin, E. J., & Nielsen, P. M. F. (2008). Bioinformatics, multiscale modeling and the IUPS Physiome Project. *Briefings in Bioinformatics*, 9(4), 333–343.
- Hunter, P. J., Li, W. W., McCulloch, A. D., & Noble, D. (2006). Multiscale modeling: physiome project standards, tools, and databases. *Computer*, 39(11), 48–54.
- Hunter, P., Robbins, P., & Noble, D. (2002). The IUPS human Physiome Project. *Pflügers Archiv: European Journal of Physiology*, 445(1), 1–9.
- Ježek, F., Kulhánek, T., Kalecký, K., & Kofránek, J. (2017). Lumped models of the cardiovascular system of various complexity. *Biocybernetics and Biomedical Engineering*, 37(4), 666–678.
- Kofránek, J., Mateják, M., & Privitzer, P. (2011). Complex model of integrated physiological systems - a theoretical basis for medical training simulators. *Mefanet Report*, 4, 22–59.
- Kofránek, J., Mateják, M., & Privitzer, P. (2011). Hummod-large scale physiological models in modelica. In *Proceedings of the 8th International Modelica Conference*; March 20th–22nd; Technical University; Dresden; Germany (pp. 713–724). Linköping University Electronic Press.
- Kofránek, J., & Rusz, J. (2010). Restoration of Guyton's diagram for regulation of the circulation as a basis for quantitative physiological model development. *Physiological Research*, 59(6), 897.
- Lloyd, C. M., Lawson, J. R., Hunter, P. J., & Nielsen, P. F. (2008). The CellML Model Repository. *Bioinformatics*, 24(18), 2122–2123.
- Maksuti, E., Bjällmark, A., & Broomé, M. (2015). Modelling the heart with the atrioventricular plane as a piston unit. *Medical Engineering & Physics*, 37(1), 87–92.
- Mangourova, V., Ringwood, J., & Van Vliet, B. (2011). Graphical simulation environments for modelling and simulation of integrative physiology. *Computer Methods and Programs in Biomedicine*, 102(3), 295–304.
- Mateják, M., & Kofránek, J. (2015). Physiome model - an integrative physiology in Modelica. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* (pp. 1464–1467).
- Mateják, M., Kulhánek, T., Šilar, J., Privitzer, P., Ježek, F., & Kofránek, J. (2014). Physiobrowser-Modelica library for physiology. In *Proceedings of the 10th International Modelica Conference*; March 10–12; 2014; Lund; Sweden (pp. 499–505). Linköping University Electronic Press.
- Mateják, M., Tribula, M., Ježek, F., & Kofránek, J. (2015). Free Modelica Library for Chemical and Electrochemical Processes. In *Proceedings of the 11th International Modelica Conference*, Versailles, France, September 21–23, 2015 (pp. 359–366). Linköping University Electronic Press.
- Ngo, C., Dahlmans, S., Vollmer, T., Misgeld, B., & Leonhardt, S. (2018). An object-oriented computational model to study cardiopulmonary hemodynamic interactions in humans. *Computer Methods and Programs in Biomedicine*, 159, 167–183.
- Nielsen, T. A., Nilsson, H., & Matheson, T. (2012). A formal mathematical framework for physiological observations, experiments and analyses. *Journal of the Royal Society Interface*, 9(70), 1040–1050.
- Omholt, S. W., & Hunter, P. J. (2016). The Human Physiome: a necessary key for the creative destruction of medicine. *Interface Focus*, 6(2), 20160003.
- Reinhardt, H. W., & Seeliger, E. (2000). Toward an Integrative Concept of Control of Total Body Sodium. *News in Physiological Sciences: An International Journal of Physiology Produced Jointly by the International Union of Physiological Sciences and the American Physiological Society*, 15, 319–325.
- Wu, K., Chen, J., Pruett, W. A., & Hester, R. L. (2013). Hummod browser: An exploratory visualization tool for the analysis of whole-body physiology simulation data. In *2013 IEEE Symposium on Biological Data Visualization (BioVis)* (pp. 97–104). ieeexplore.ieee.org.

The OpenModelica Integrated Modeling, Simulation and Optimization Environment

Peter Fritzson¹, Adrian Pop¹, Adeel Asghar¹, Bernhard Bachmann¹, Willi Braun², Robert Braun¹, Lena Buffoni¹, Francesco Casella³, Rodrigo Castro⁶, Alejandro Danós⁶, Rüdiger Franke⁷, Mahder Gebremedhin¹, Bernt Lie⁸, Alachew Mengist¹, Kannan Moudgalya⁵, Lennart Ochel¹, Arunkumar Palanisamy¹, Wladimir Schamai⁹, Martin Sjölund¹, Bernhard Thiele¹, Volker Waurich⁴, Per Östlund¹

¹PELAB – Programming Environment Lab, Dept. of Computer and Information Science
Linköping University, SE-581 83 Linköping, Sweden

²FH Bielefeld, Bielefeld, Germany

³Dept. Electronics and Information, Politecnico di Milano, Milan, Italy

⁴TU Dresden, Dresden, Germany

⁵IIT Bombay, Mumbai, India

⁶Dept. Computer Science, Universidad de Buenos Aires, Argentina

⁷ABB AG, DE-68309 Mannheim, Germany

⁸University of South-Eastern Norway, Porsgrunn, Norway

⁹Danfoss Power Solutions GmbH & Co. OHG, Offenbach, Germany

peter.fritzson@liu.se, adrian.pop@liu.se

Abstract

OpenModelica is currently the most complete open-source Modelica- and FMI-based modeling, simulation, optimization, and model-based development environment. Moreover, the OpenModelica environment provides a number of facilities such as debugging; optimization; visualization and 3D animation; web-based model editing and simulation; scripting from Modelica, Python, Julia, and Matlab; efficient simulation and co-simulation of FMI-based models; compilation for embedded systems; Modelica-UML integration; requirement verification; and generation of parallel code for multi-ore architectures. The environment is based on Modelica and uses an extended version of Modelica for its implementation. This overview paper intends to give an up-to-date brief description of the capabilities of the system, and the main vision behind its development.

Keywords: Modelica, OpenModelica, MetaModelica, FMI, modeling, simulation, optimization, development, environment, compilation, embedded system, real-time

1 Introduction

The OpenModelica environment was the first open source Modelica environment supporting the Modelica modeling language (Modelica Association 2017) (Fritzson 2014). Its development started in 1997 resulting in the release of a flattening frontend for a core subset of Modelica 1.0 in 1998. After a pause of four years, the open source development resumed in 2002. An early version of OpenModelica is described in (Fritzson et al 2005). Since then the capabilities of

OpenModelica have expanded enormously. The Open Source Modelica Consortium which supports the long-term development of OpenModelica was created in 2007, initially with seven founding organizations. The scope and intensity of the open source development has gradually increased. At the time of this writing the consortium has fifty-three supporting organizational members. The long-term vision for OpenModelica is an integrated and modular modeling, simulation, model-based development environment with additional capabilities such as optimization, sensitivity analysis, requirement verification, etc., which are described in the rest of this paper. The previous overview paper about OpenModelica was published 2005. The current paper intends to give a more up-to-date overview of the system and the vision and goals behind its development.

This paper is organized as follows. Section 2 presents the idea of integrated environment, Section 3 the goals for OpenModelica, Section 4 an overview of the OpenModelica environment, Section 5 and its subsections give more details about OpenModelica and its subsystems, Section 6 presents related work and Section 7 the conclusions.

2 Integrated Interactive Modeling and Simulation Environments

An integrated interactive modeling and simulation environment is a special case of programming environments with applications in modeling and simulation. Thus, it should fulfill the requirements both from general integrated interactive environments and

from the application area of modeling and simulation mentioned in the previous section.

The main idea of an integrated programming environment in general is that a number of programming support functions should be available within the same tool in a well-integrated way. This means that the functions should operate on the same data and program representations, exchange information when necessary, resulting in an environment that is both powerful and easy to use. An environment is interactive and incremental if it gives quick feedback, e.g., without re-computing everything from scratch, and maintains a dialogue with the user, including preserving the state of previous interactions with the user. Interactive environments are typically both more productive and more fun to use than non-interactive ones.

There are many things that one wants a programming environment to do for the programmer or modeler, particularly if it is interactive. Comprehensive software development environments are expected to provide support for the major development phases, such as:

- Requirements analysis
- Design
- Implementation
- Maintenance

A pure programming environment can be somewhat more restrictive and need not necessarily support early phases such as requirements analysis, but it is an advantage if such facilities are also included. The main point is to provide as much computer support as possible for different aspects of systems development, to free the developer from mundane tasks so that more time and effort can be spent on the essential issues.

Our vision for an integrated interactive modeling and simulation environment is to fulfill essentially all the requirements for general integrated interactive environments combined with the specific needs for modeling and simulation environments, e.g.:

- Specification of requirements, expressed as documentation and/or mathematics
- Design of the mathematical model
- Symbolic transformations of the mathematical model
- A uniform general language for model design, mathematics, and transformations
- Automatic generation of efficient simulation code
- Execution of simulations
- Debugging of models
- Design optimization
- Evaluation and documentation of numerical experiments
- Graphical presentation

- Model and system structure parameterization
- Variant and version handling, traceability

3 Goals for OpenModelica

The computational and simulation goals of the OpenModelica tool development include, but are not limited to, the following:

- Providing a complete open source Modelica-based industrial-strength implementation of the Modelica language, including modeling and simulation of equation-based models, system optimization, and additional facilities in the programming/modeling environment.
- Providing an interactive computational environment for the Modelica language. It turns out that with support of appropriate tools and libraries, Modelica is very well suited as a computational language for development and execution of numerical algorithms, e.g. for control system design and for solving nonlinear equation systems.

The research related goals and issues of the OpenModelica open source implementation of a Modelica environment include, but are not limited to, the following:

- Development of a *complete formal specification and reference implementation* of Modelica, including both static and dynamic semantics. Such a specification can be used to assist current and future Modelica implementers by providing a semantic reference, as a kind of reference implementation.
- *Language design*, e.g. to further *extend the scope* of the language, e.g. for use in diagnosis, structural analysis, system identification, integrated product development with requirement verification, etc., as well as modeling problems that require partial differential equations.
- *Language design to improve abstract properties* such as expressiveness, orthogonality, declarativity, reuse, configurability, architectural properties, etc.
- *Improved implementation techniques*, e.g. to enhance the performance of compiled Modelica code by generating code for parallel hardware.
- *Improved debugging* support for equation based languages such as Modelica, to make them even easier to use.
- *Improved optimization support*, with integrated optimization and modeling/simulation. Two kinds: parameter-sweep optimization based on multiple simulations; direct *dynamic optimization* of a goal function without lots of simulations, e.g., using collocation or multiple shooting.
- *Easy-to-use* specialized high-level (graphical) *user interfaces* for certain application domains.

- *Visualization* and animation techniques for interpretation and presentation of results.
- *Integrated requirement modeling and verification support*. This includes the ability to enter requirements formalized in a kind of Modelica style, and to verify that the requirements are fulfilled for selected models under certain usage scenarios.

The OpenModelica effort started by developing a rather complete formal specification of the Modelica language. This specification was developed in *Operational Semantics*, which still is the most popular and widely used semantics specification formalism in the programming language community. It was initially used as input for automatic generation of the Modelica translator implementations which are part of the OpenModelica environment. The RML compiler generation tool (our implementation of Operational Semantics) (Fritzson et al, 2009) was used for this task.

However, inspired by our vision of integrated interactive environments with self-specification of programs and data, and integrated modeling and simulation environments), in 2005 we designed and implemented an extension to Modelica called MetaModelica (Pop and Fritzson, 2006), (Fritzson, Pop, Sjölund, 2011). This was done in order to support language modeling and specification (including modeling the language itself), in addition to the usual physical systems modeling applications of Modelica, as well as applications requiring combined symbolic-numeric capabilities. Modeling the semantics in itself was also inspired by functional languages such as Standard ML (Milner 1997), and OCaml (OCaml org, 2018). Moreover, it was an investment into a future Modelica becoming a combined symbolic-numeric language such as Mathematica, but more efficient and statically strongly typed.

This language extension has a backwards-compatible Modelica-style syntax but was initially implemented on top of the RML compiler kernel. The declarative specification language primitives in RML with single-assignment pattern equations, possibly recursive case records (in MetaModelica called uniontypes) and match expressions, fit well into Modelica since it is a declarative equation-based language. In 2006 our whole formal specification of Modelica static and translational semantics, at that time about 50 000 lines, was automatically translated into MetaModelica. After that, all further development of the symbolic processing parts of the OpenModelica compiler (the run-time parts were mainly written in C), was done in MetaModelica.

At the same time we embarked on an effort to completely integrate the MetaModelica language extension into the Modelica language and the OpenModelica compiler. This would enable us to support both Modelica and MetaModelica by the same compiler. This would allow modeling the Modelica tool and the OpenModelica compiler using its own language.

This would get rid of the limitations of the RML compiler kernel and the need to support two compilers. Moreover, additional tools such as our Modelica debugger can be based on a single compiler.

Such an ability of a compiler to compile itself is called compiler *bootstrapping*. This development turned out to be more difficult and time-consuming than initially expected; moreover, developers were not available for a few years due resource limitations and other priorities. Finally, bootstrapping of the whole OpenModelica compiler was achieved in 2011. Two years later, in 2013, all our OpenModelica compiler development was shifted to the new bootstrapped compiler (Sjölund, Fritzson, Pop, 2014), (Sjölund, 2015), after automatic memory reclamation (garbage collection), separate compilation, and a new efficient debugger had been achieved for our new compiler platform.

More recently, we have had an effort to restructure and rewrite the frontend part of the OpenModelica compiler (OMC). The reasons were two-fold: to support the exact Modelica semantics required to simulate certain models even though the semantics at that time was not clearly specified by the Modelica language specification (Modelica Association 2017), and to achieve much higher compilation speed for large and complex models. This work turned out to more difficult than expected. Fortunately, recently, a lot of progress has been made and a release of a preliminary version of this new frontend as part of OMC now appears feasible late fall 2018.

4 The OpenModelica Environment

At the time of this writing, the interactive OpenModelica environment primarily consists of the following components and subsystems:

- *A graphical and textual model editor*, OMEdit. This is a graphical connection editor for component based model design by connecting instances of Modelica classes. The editor also provides text editing. Moreover, the OMEdit GUI provides a graphical user interface to simulation and plotting (OMPlot). See Section 5.2.
- *An interactive session handler*, OMShell, that parses and interprets commands and Modelica expressions for evaluation. The session handler also contains simple history facilities, and completion of file names and certain identifiers in commands. There is also a Python variant of the interactive session handler called OMPython that supports the same commands in Python. Very recently, similar session handlers for Julia, called OMJulia, and Matlab, called OMMatlab, have been implemented. See Section 5.10.
- *A Modelica compiler*, OMC, translating Modelica to lower level code such as C code, with a symbol table

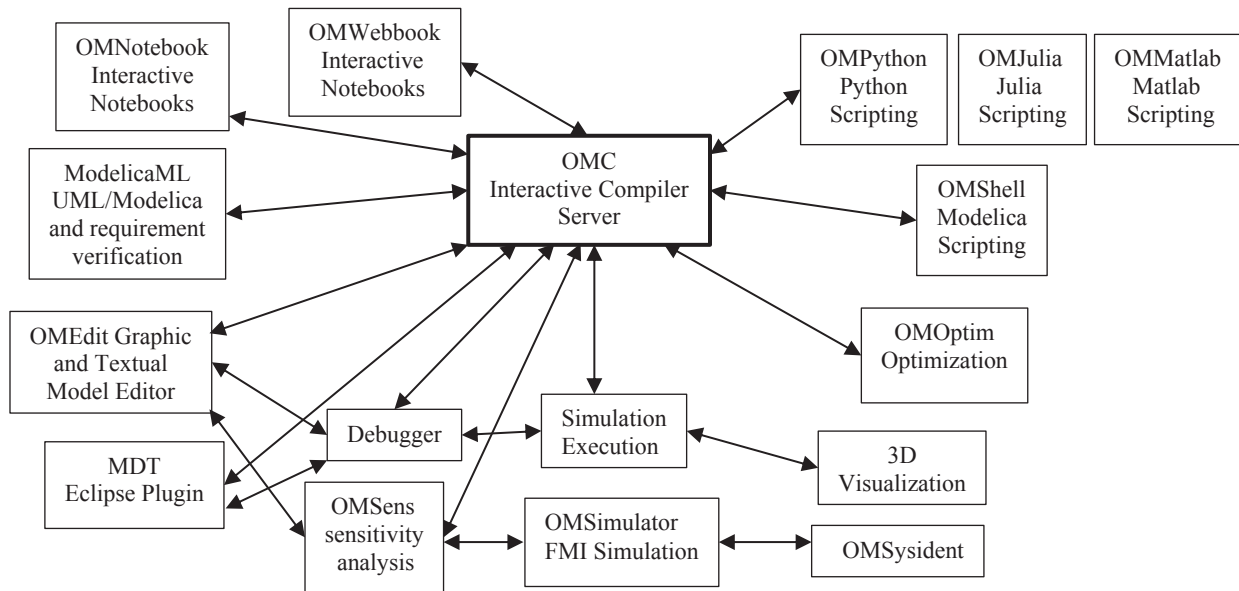


Figure 1. The architecture of the OpenModelica environment. Arrows denote data and control flow.

containing definitions of models, functions, and variables. Such definitions can be predefined, user-defined, or obtained from libraries. See Section 5.1. There is also a compilation mode to generate low-footprint code for embedded systems, see Section 5.13.

- *An execution and run-time module.* This module currently executes compiled binary code from translated models and functions. It includes numerical solvers as well as event handling facilities for the discrete and hybrid parts of the Modelica language. See Section 5.6.
- *Debuggers and performance analyzers.* These tools provide source-level Modelica debugging on equation models (Section 5.5), algorithmic model code (Section 5.4), as well as performance analysis of models (Section 5.5).
- *Textual model editors.* Any text editor can be used. Among the OpenModelica tools, text editing of models is supported by OMEdit (Section 5.2), by the OpenModelica MDT Eclipse plug-in (Section 5.6), and by the interactive electronic book OMNotebook (Section 5.7).
- *An interactive electronic book, OMNotebook.* This tool provides an active electronic book facility supporting chapters, sections, execution of simulation models, plotting, etc. One book, DrModelica, for teaching Modelica to the beginner, is automatically opened by default. The user can define his/her own books. This tool is useful for developing interactive course material. See Section 5.7.
- *Jupyter notebook for OpenModelica.* More recently, the Python-based Jupyter notebook has appeared, supporting a number of languages. Therefore we

have also developed a Jupyter notebook for OpenModelica (OSMC 2018a) using Modelica scripting. However, Python scripting together with the OMPython package is used in the Jupyter notebooks presented in (Lie et al, 2016)

- *An interactive web-based electronic book, OMWebbook.* This is similar to OMNotebook, but model editing and simulation is in a web-browser. Simulation is performed by a simulation server. See Section 5.8.
- *An optimization module using parameter sweeps, called OMOptim.* This tool performs optimizations by running several simulations for different parameter settings while searching for the optimum value of a user-specified goal function. See Section 5.17.
- *A dynamic optimization module.* Direct optimization (without running lots of simulations) of a whole solution trajectory using collocation or multiple shooting. A goal function can be formulated to be optimized under the constraints of a selected model. See Section 5.17.
- *Requirement verification and ModelicaML Eclipse plug-in.* This plug-in contains a Modelica-UML profile that allows integrated requirement verification and cyber-physical hardware-software modeling by combining hardware modeling in Modelica with software modeling using UML. The tool contains a UML to Modelica translator that makes it possible to simulate combined UML-Modelica models. Moreover, automatic (dynamic) verification of formalized requirements against selected scenarios is supported by ModelicaML or by a Modelica-based approach without using UML. See Section 5.15 and Section 5.16.

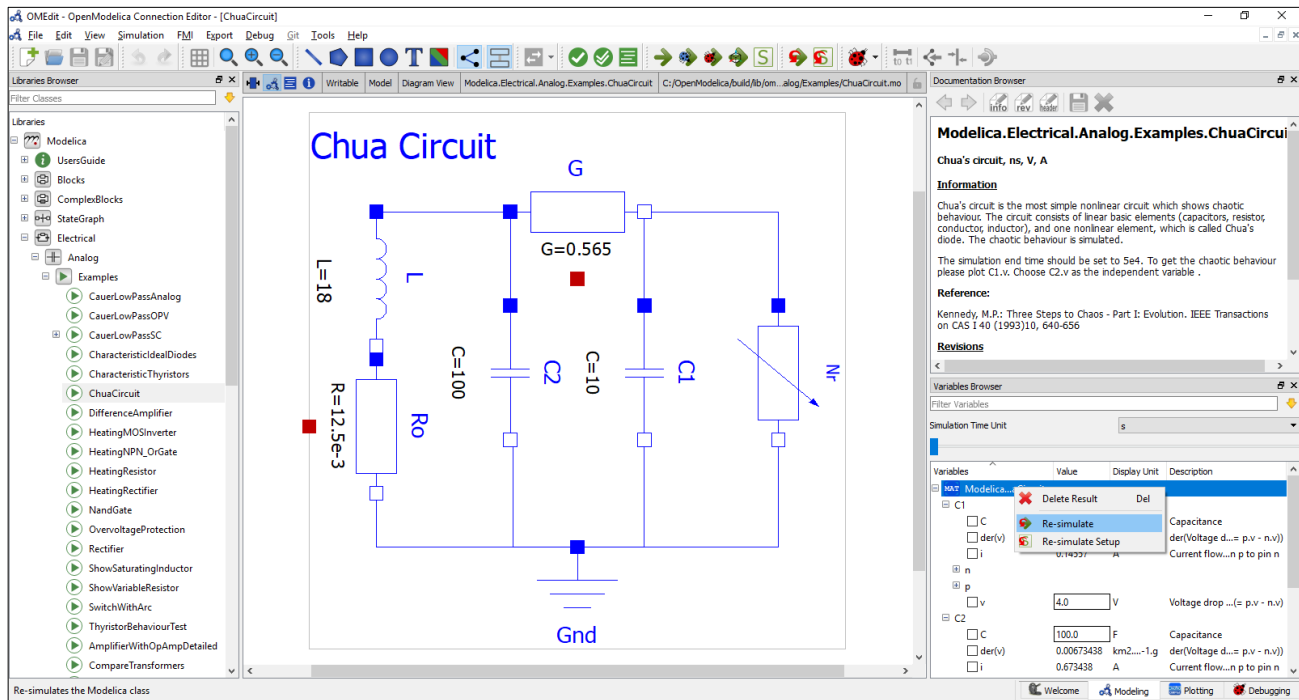


Figure 2. OMEdit on the Modelica.Electrical.Analog.Examples.ChuaCircuit model. *Center:* Model connection diagram. *Upper right:* information window. *Lower right:* plot variable browser will a small popup re-simulate menu on top.

- *MDT Eclipse plug-in.* The MDT (Modelica Development Tooling) Eclipse plug-in for Modelica library and model compiler textual development, project support, cross-referencing, building executables, debugging, etc. See Section 5.6.
- *3D animation visualization.* This is provided by a special module in OpenModelica, and uses the standard Modelica MBS library 3D graphical annotations. See Section 5.3.
- *FMI Import and Export.* A model (including models from other tools, even non-Modelica ones) can be imported or exported according to the FMI (Functional Mockup Interface) standard as an FMU (Functional Mockup Unit).
- *OMSimulator* FMI-based simulation and co-simulation subsystem. This recently added subsystem, which also can be run stand-alone separated from the OpenModelica compiler, supports efficient simulation and co-simulation of single or composite FMUs. FMUs can also be connected using a graphical editor to form composite FMUs. See Section 5.12.
- *OMSens.* Sensitivity analysis subsystem that allows both single-parameter and multi-parameter analysis, the latter based on robust optimization techniques. Specification of the analysis and display of results can be made interactively via OMEdit in the current prototype. An early prototype not yet integrated in OMEdit is described in (Danós et al, 2017).
- *OMSysIdent.* A parameter system identification module, using system identification vs

measurement data to determine the best model parameter values for a certain model (OSMC 2018c).

- *MetaModelica language extension.* This is used for modeling/specification of languages (including the Modelica language) and for Modelica programming of model transformations (Pop and Fritzson, 2006), (Fritzson, Pop, Sjölund, 2011). Related to this, there are discussions in the Modelica Design group about possible extensions to the Modelica language that would enable definition some language constructs in a Modelica core library instead of being hardcoded in the compiler.
- *Parallelization and ParModelica language extension.* ParModelica is used for explicit algorithmic parallel Modelica programming with compilation to both multi-core CPU platforms and GPGPU platforms (including NVIDIA). See Section 5.18.

5 OpenModelica Subsystems

The relationships between the main OpenModelica subsystems is depicted above in Figure 1. Their functionality is briefly described in the following.

5.1 OMC – The OpenModelica Model Compiler

OMC is the OpenModelica compiler which translates Modelica models into C/C++ code (or Java or C# code using experimental code generators), which is compiled and executed to perform simulations. The OpenModelica compiler is generated from formal specifications in RML (earlier) or MetaModelica (currently). At the time of this writing the OpenModelica compiler (OMC) is generated from a specification of about two hundred thousand lines of MetaModelica. Moreover, OMC is able to compile itself, i.e., it is bootstrapped.

5.2 OMEdit – the OpenModelica Graphic Model Editor and Simulator GUI

OMEdit is the OpenModelica graphic model editor (Figure 2), (Asghar et al, 2011). In addition to graphic/textual model editing and browsing, it also provides model text editing, simulation, parameter update, debugging, and plotting capabilities.

Using OMEdit to perform simulations and plotting simulation results is depicted in Figure 3 below.

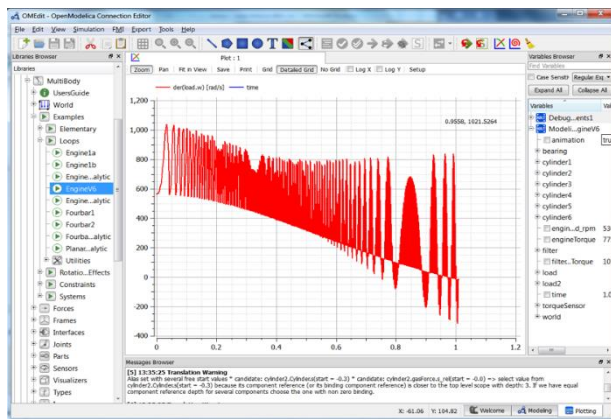


Figure 3. OpenModelica simulation of the V6Engine model with 11000 equations. Plotting simulation results using OMEdit. *Left*: Model browser. *Right*: Plot variable browser. *Bottom*: message browser window.

5.3 3D Animation and Visualization

The OpenModelica 3D animation and visualization is a built-in feature of OMEdit to animate based on 3D shapes defined by the MSL Multi-Body library. It provides visualization of simulation results and animation of geometric primitives and CAD-files. There is also support for FMI-based visualization (Waurich and Weber, 2017).

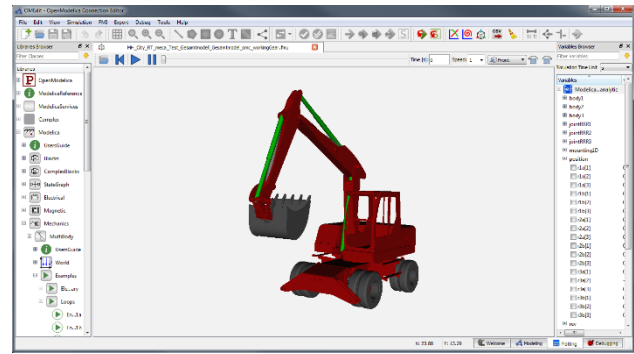


Figure 4. OpenModelica 3D animation of a simulated excavator.

5.4 The OpenModelica Algorithm Debugger

The OpenModelica algorithm debugger (Figure 5), (Pop, 2008), (Sjölund, 2015) is available for use either from OMEdit or from the MDT Eclipse plug-in. The debugger provides traditional debugging of the algorithmic part of Modelica, such as setting breakpoints, starting and stopping execution, single-stepping, inspecting and changing variables, inspecting all kinds of standard Modelica data structures as well as MetaModelica data structures such as trees and lists.

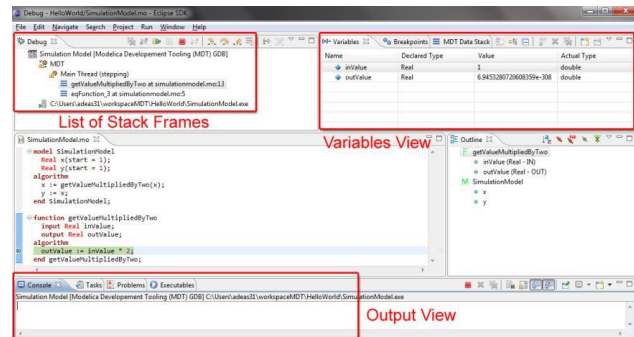


Figure 5. The OpenModelica algorithmic code debugger viewed from the MDT Eclipse plug-in. The OMEdit version of the debugger looks about the same. A breakpoint has been set in the function which is called from the small model called SimulationModel.

5.5 The OpenModelica Equation Model Debugger and Performance Analyzer

The OpenModelica equational model debugger (Figure 6), (Pop, Sjölund, et al, 2014), (Sjölund, 2015) is available for use from OMEdit. It provides capabilities for debugging equation-based models, such as showing and explaining the symbolic transformations performed on selected equations on the way to executable simulation code. It can locate the source code position of an equation causing a problem such as a run-time error, traced backwards via the symbolic transformations. Moreover, a *performance analyzer tool* is also included in OpenModelica and integrated with the debugger.

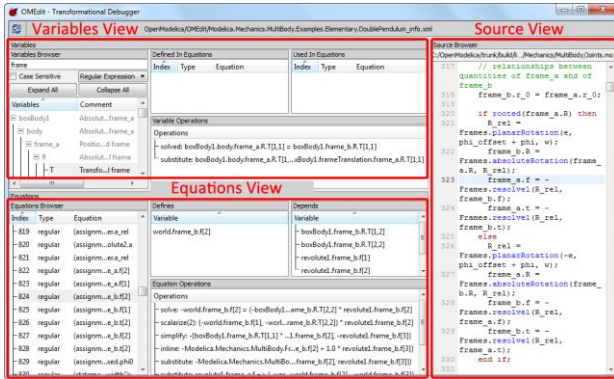


Figure 6. The OpenModelica equation model debugger. *Left:* equations view where equations and symbolic transformations can be viewed. *Right:* source view where the erroneous equation is pointed out.

5.6 Run-time Solver Module and DAEMode

The OpenModelica execution and run-time solver module executes compiled binary code from translated models and functions. It includes numerical solvers as well as event handling facilities for the discrete and hybrid parts of the Modelica language.

A recent extension of this module is the DAEMode used for solving very large models. This is part of an emerging trend in Modelica tools of handling large-scale models, with hundreds of thousands or possibly millions of equations, (Casella, 2015). OpenModelica has pioneered this field by introducing sparse solvers in the solution chain: KLU for linear algebraic equations, Kinsol for nonlinear algebraic equations, and IDA for causalized differential equations. It also introduced the direct use of IDA as differential-algebraic equation solver, skipping the traditional causalization step, which is computationally more efficient for certain classes of systems. The largest system handled so far is an electro-mechanical power system model with about 600.000 differential-algebraic equations, (Braun et al, 2017).

5.7 OMNotebook and DrModelica

OMNotebook (Figure 7) (Fernström et al, 2006) is a book-like interactive user interface to OpenModelica primarily intended for teaching and course material. It supports sections and subsections to any level, hiding and showing sections and cells, interactive evaluation and simulation of Modelica models and plotting results. The DrModelica (Lengquist-Sandelin, 2003) interactive Modelica teaching course was the first main application, at that time based on Mathematica notebooks.

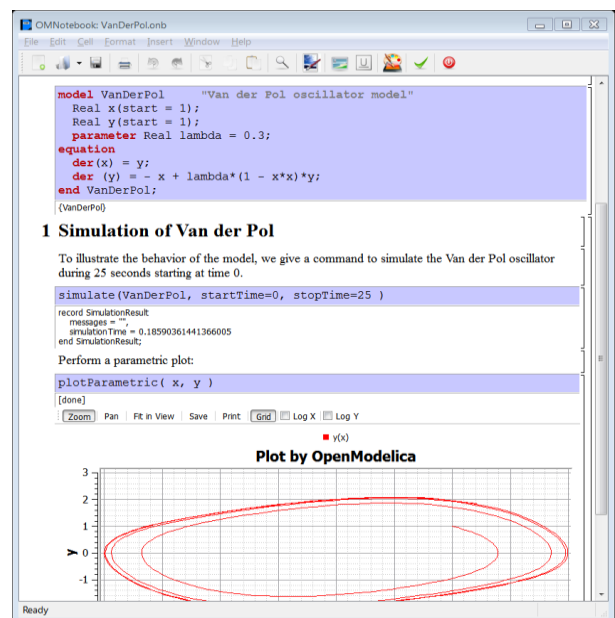
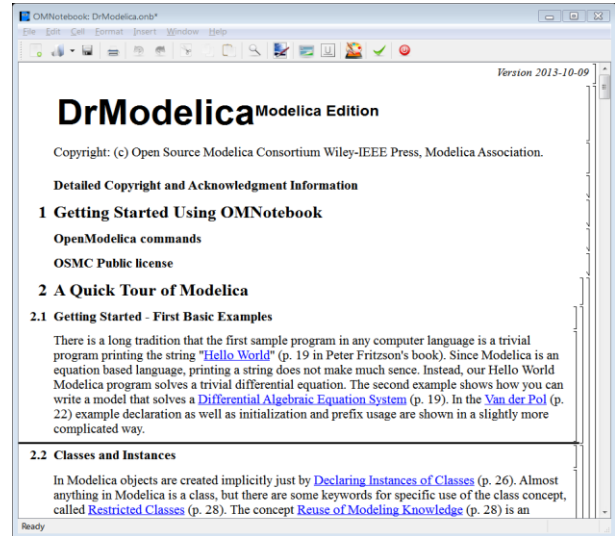


Figure 7. The OMNotebook electronic notebook showing part of the DrModelica document (course-material) for learning Modelica. *Top:* The DrModelica document start page. *Bottom:* The VanDerPol sub-document showing a cell with a Modelica model, simulation commands, and plot results.

5.8 OMWebbook – Interactive Web-based Editable and Executable Book

OMWebbook (Figure 8) (Moudgalya et al, 2017), (Fritzson et al, 2018), is an interactive web-based electronic book. This is similar to OMNotebook, but textual model editing and simulation is performed in a web-browser. Simulation is performed by a dedicated simulation server. Thus, the user need not install OpenModelica on a computer. Editing and simulation can even be done from smartphones or tablets.

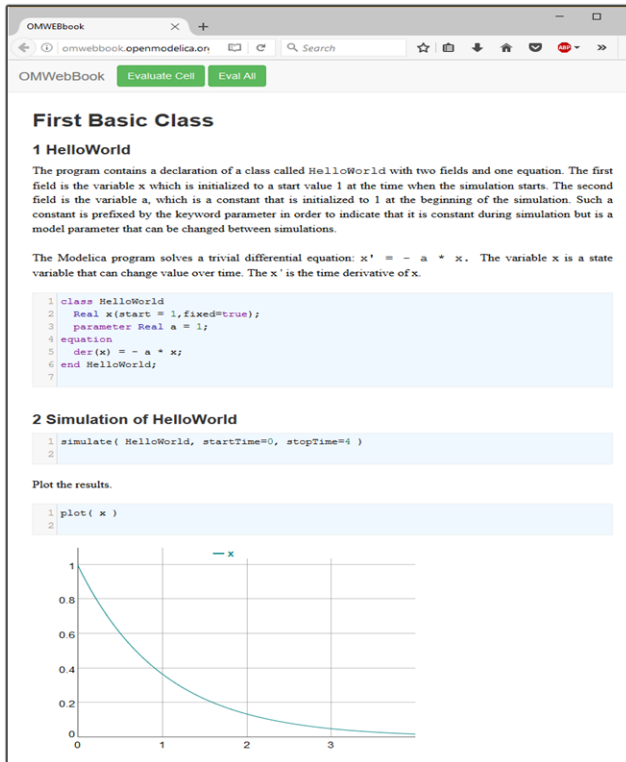


Figure 8. OMWebbook with editable models, simulations, and plots.

5.9 MDT Eclipse Plug-in

The MDT (Modelica Development Tooling) Eclipse plug-in (Figure 9) (Pop et al, 2006), (Pop 2008), is an Eclipse-based textual development environment for Modelica and MetaModelica model development.

It provides the usual facilities for software development such as browsing, building, cross referencing, syntax checking, and showing useful information such as types, function signatures, etc.

MDT is primarily used for development of medium to large scale Modelica projects, such as Modelica libraries written in standard Modelica and the OpenModelica compiler (currently containing more than 200 packages) written in MetaModelica.

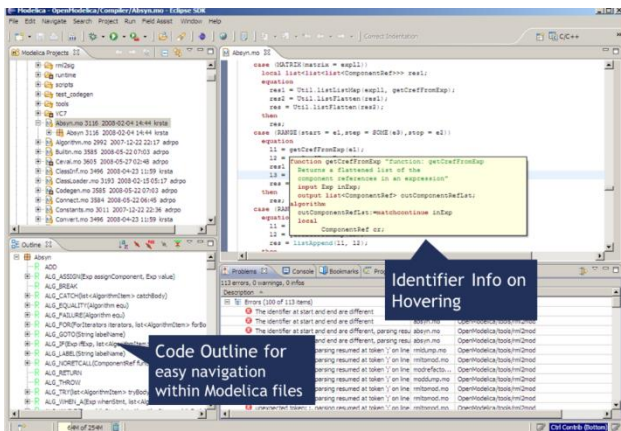


Figure 9. The OpenModelica MDT (Modelica Development Tooling) Eclipse plug-in.

5.10 Python, Julia, and Matlab Scripting

Scripting APIs to OpenModelica is also provided for the languages Python (Python 2018), Julia (Julia org, 2018), and Matlab (MathWorks 2018), through the subsystems OMPython (Lie et al, 2016), OMJulia and OMMatlab (OSMC 2018a). This gives the user the possibility to use Modelica together with the rich set of facilities and libraries in these languages, e.g. for tasks such as control design and post processing of simulation results.

5.11 Spoken Tutorials for OpenModelica

A number of interactive audio-video based spoken tutorials (www.spoken-tutorial.org) have been developed which provide step-by-step teaching about how to use OpenModelica and develop simple models. (Moudgalya et al, 2017). They are not part of the OpenModelica installer and system, but mentioned here since they provide important functionality to learn OpenModelica and Modelica.

5.12 OMSimulator – FMI-based Simulation and Composite Model Editor

OMSimulator, version 2.0, is an OpenModelica subsystem that provides efficient simulation and co-simulation of models in the (pre-compiled) FMI standard FMU (Functional Mockup Unit) form. Thus, models from non-Modelica tools compiled into FMUs can also be included and simulated. Furthermore, models that cannot be exported as FMUs can be integrated in a simulation using tool-to-tool co-simulation. This is provided via wrappers to models in tools such as ADAMS, Beast, Simulink, Hopsan, or co-simulation of FMUs with embedded solvers. The system can optionally be used with TLM (Transmission Line Modeling) connectors, which give numerically more stable co-simulation.

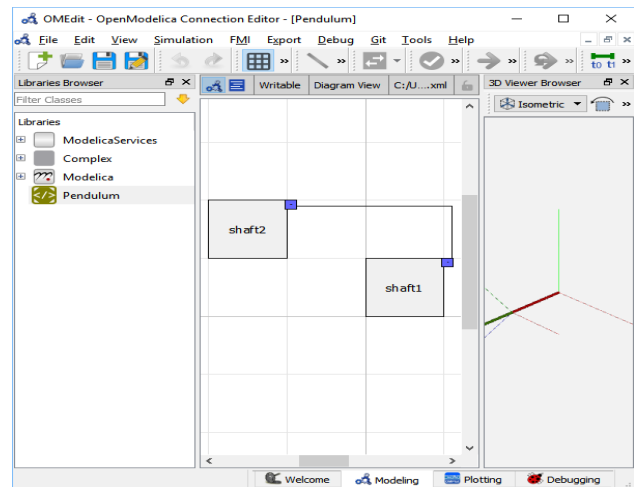


Figure 10. The OpenModelica OMSimulator composite model editor including 3D animation.

The earlier version OMSimulator 1.0, was already available in OpenModelica 1.12.0 released 2017

(Fritzson, Braun, and Hartford, 2018), (OSMC 2018b), but in that version TLM-connectors were mandatory and pure FMI model-exchange based simulation was missing.

Moreover, OMSimulator contains a composite model editor integrated in OMEdit, that allows combining external models (FMUs, both model-exchanged and co-simulated ones) into new composite models.

This editor, an extension of OMEdit (Figure 10), also provides 3D visualization of connected mechanical model components which can be FMUs, Modelica models, etc., or co-simulated components. 3D animation of simulated FMUs is possible (right part of Figure 10). A composite model is saved as an XML file according to the SSP (Systems and Structure Parameterization) standard (Modelica Association 2018), (OSMC 2018c).

5.13 Embedded System Support

OpenModelica provides code generation of real-time controllers from Modelica models, for small foot-print platforms such as Arduino boards. (Berger et al, 2017), or in tools for RexRoth PLCs (Menager et al, 2014).

One example of code generation to small targets is the Single board heating system (Figure 11) from IIT Bombay (Arora, Kannan Moudgalya, and Malewar, 2010). It is used for teaching basic control theory, and usually controlled by a serial port (set fan value, read temperature, etc). OpenModelica can generate code targeting the ATmega16 on the board.

The program size is 4090 bytes including LCD driver and PID-controller (out of 16 kB flash memory available). The ATmega16 we target has 1 kB SRAM available for data (stack, heap, and global variables). In this case, only 130 bytes is used for data variables.



Figure 11. The SBHS (Single Board Heating System), an example embedded target system for OpenModelica.

To simplify interfacing of low-level devices from Modelica, OpenModelica supports the Modelica DeviceDrivers library (Thiele, Beutlich, Waurich, Sjölund, and Bellmann, 2017), which is a free library for interfacing hardware drivers that is developed primarily for interactive real-time simulations. It is cross-platform (Windows and Linux). Using this library, modeling, parameterization and configuration can be done at a high level of abstraction using

Modelica, avoiding the need for low level C programming.

5.14 Model-based Control, Synchronous Modelica, and C++ Run-time

OpenModelica is one of the (currently) two Modelica tools that support synchronous Modelica (Modelica Association, 2017), implemented both on top of the OpenModelica C run-time and C++ run-time. This can be used for model-based control, using Modelica and FMI, (Franke et al, 2017), and using the OpenModelica C++ run-time (Franke et al, 2015).

5.15 ModelicaML Modelica-UML Profile and Eclipse Plug-in

ModelicaML (Figure 12), (Schamai, 2013), (Schamai et al, 2014) is an Eclipse plug-in and Modelica-UML profile for the description of system architecture and system dynamic behavior. It is based on an extended subset of the OMG Unified Modeling Language (UML) as well as Modelica, and is designed for Modelica code generation from graphical models such as state machines and activity diagrams, supporting hardware/software co-modeling and system requirement verification against selected scenarios. The current prototype has not been updated recently and only works together with an old version of Eclipse.

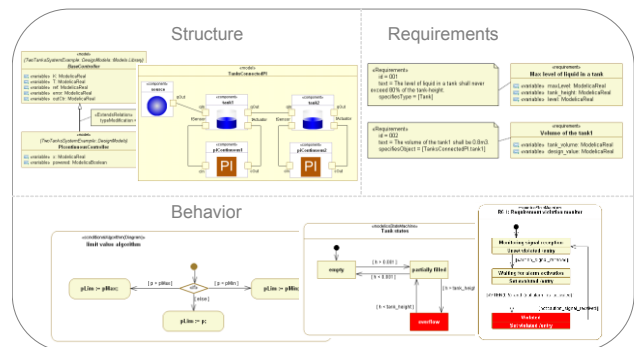


Figure 12. The ModelicaML Eclipse plug-in and UML-Modelica profile for integrated software-hardware modeling and requirements verification.

5.16 Requirement Verification

OpenModelica supports requirement verification using the vVDR approach (virtual Verification of Designs vs. Requirements), (Schamai, 2014<, Schamai et al, 2015). It was first introduced in the ModelicaML Eclipse plug-in mentioned previously, using a combination of UML and Modelica for requirement specification. Recently, a Modelica-only version of vVDR has been designed and implemented in OpenModelica, using requirement specification in Modelica, and a vVDR Modelica library (Buffoni et al, 2014; Buffoni et al, 2017).

It is a simulation-based approach that can be used to verify (Figure 13) different design alternatives against sets of requirements using different scenarios. The tool

automatically generates verification models in Modelica, performs the simulations, compares the results, and generates a report about verification results.

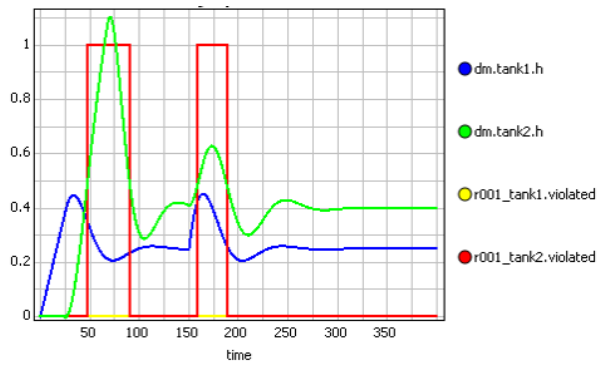


Figure 13. Simulation-based requirement verification for the two-tanks example. Requirement r001 regarding the level of tank2 is violated twice (shown in red).

5.17 Design Optimization

Two forms of design optimization tool support are available with OpenModelica: a) the traditional parameter sweep static design optimization using many simulation runs, or b) direct dynamic optimization of a full trajectory. The first method is supported by the OMOptim tool (Figure 14), (Thieriot et al, 2011).

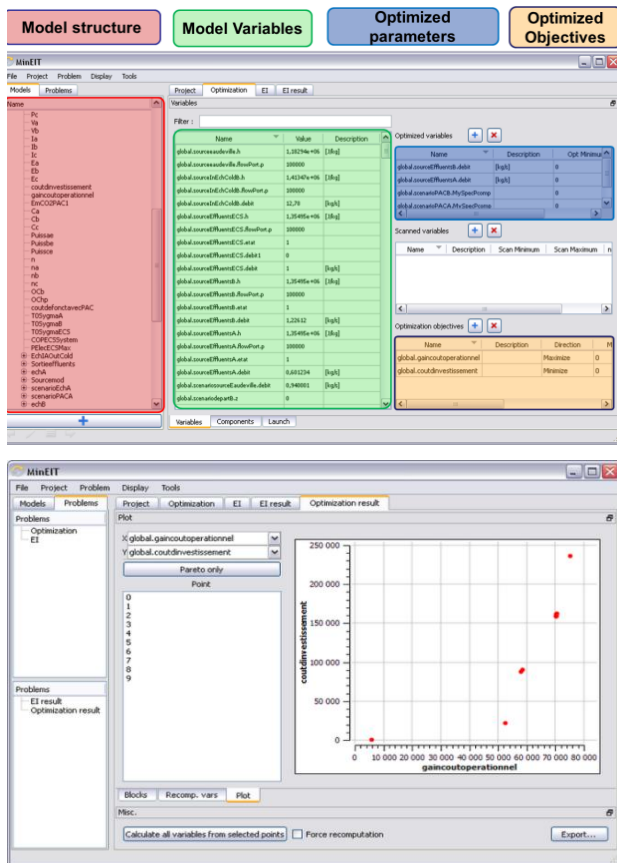


Figure 14. The OpenModelica OMOptim tool for parameter sweep optimization. *Top:* selecting variables, objectives, parameters. *Bottom:* A result plot with a Pareto optimization of two goal functions.

The second approach, dynamic optimization (Figure 15), (Bachmann et al, 2012), (Åkesson, 2008), formulates an optimization problem directly on a whole trajectory which is divided into trajectory segments (Figure 15, top) whose shapes are determined by coefficients which are initially not determined.

During the optimization process these coefficients are gradually assigned values which make the trajectory segments adjust shape and join into a single trajectory with a shape that optimizes the goal function under the constraints of fulfilling the model equations. Figure 15 (bottom) shows the relative speedup of performing dynamic optimization of a goal function for a small BatchReactor model using parallel versions of the dynamic optimization methods multiple shooting and multiple collocation running on a multi-core computer. Optimization algorithms from the Ipopt library (Wächter and Biegler, 2006), are employed for part of the optimization mechanism.

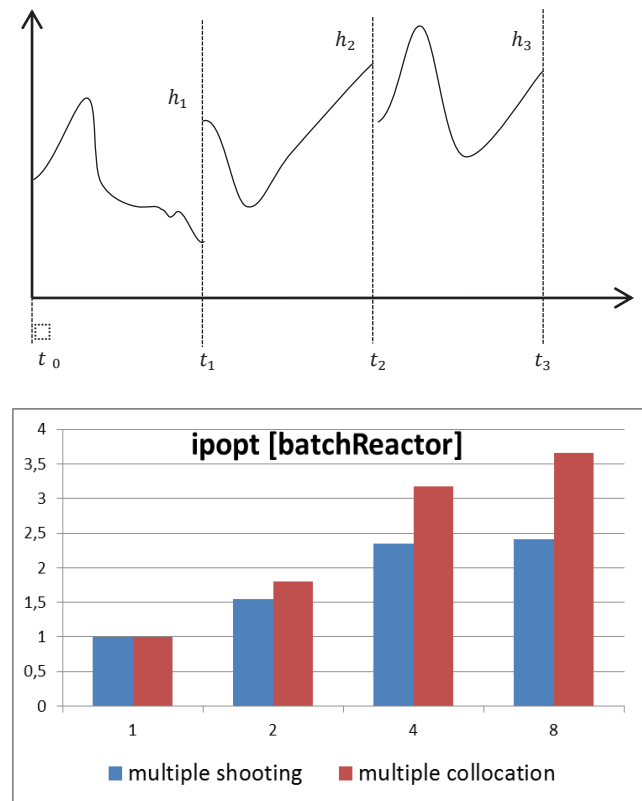


Figure 15. *Top:* Dynamic optimization formulates the whole trajectory in terms of trajectory segments whose shapes are adjusted during optimization. *Bottom:* Relative speedups and computation times of the complete dynamic optimization process for the BatchReactor example model using parallel multiple shooting or multiple collocation in OpenModelica on 1, 2, 4, and 8 cores.

5.18 Parallelization and Multi-Core

Work on generating parallel code from Modelica models has been ongoing for OpenModelica during several years. Automatic extraction of task parallelism and automatic scheduling is one approach that has been

investigated (Figure 16), (Aronsson 2006), (Walther et al, 2014). Another approach is the ParModelica language extension (Gebremedhin 2011) that allows generation of OpenCL code for data-parallel platforms such as the NVIDIA graphics cards. A third approach, which is now integrated with the above approaches is dynamic load balancing partly based on the running simulation. (Gebremedhin and Fritzson, 2017).

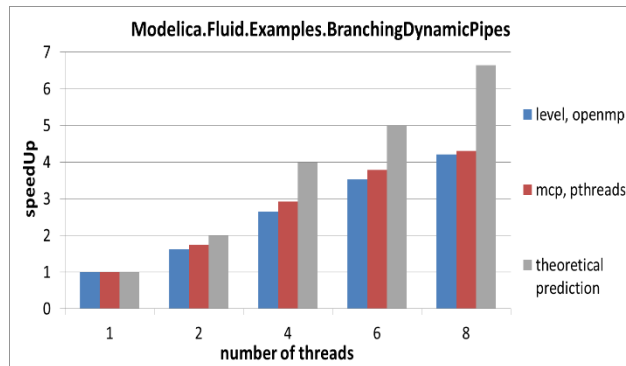


Figure 16. Example of speedup of parallel code from OpenModelica for the Fluid.Examples.Branching.DynamicPipes model.

6 Related Work

Since OpenModelica is a Modelica environment it has of course been influenced by other Modelica tools. The most influential of these tools is Dymola (Elmqvist et al, 1996), (Brück et al, 2002), (Dassault Systèmes 2013), which was the first full-scale industrial-strength Modelica environment. Certain aspects have also been influenced by the MathModelica environment (Fritzson 2006), later renamed and further developed to Wolfram System Modeler (Wolfram Research 2018), InterLisp, Mathematica (Wolfram 2003), and ObjectMath (Fritzson, et al, 1995) have influenced the design of OpenModelica as an integrated symbolic-numeric environment. Recently, the rapidly developing symbolic-numeric Julia language (Bezanson 2017), (Julia org, 2018) has appeared, with similar goals as MetaModelica.

7 Conclusions

OpenModelica has been developed into a powerful open source tool suite for modeling, simulation, and model-based development. Still some challenges are being worked on and remain to be addressed, for example very large models with several million equations. The debugger can be further improved to find additional kinds of numeric/symbolic errors. Integration aspects between tool functionalities can be further enhanced. Just-in-time compilation would improve the system's interactive properties. Two large recent OpenModelica efforts are the OMC new frontend development for 100% coverage and greatly enhanced compilation speed, and the OMSimulator tool for efficient large-

scale FMI-based simulation. Recently OMJulia has been introduced that provides OpenModelica access from Julia. More powerful integration options between Julia and OpenModelica are also being considered in order to benefit from the Julia libraries and infrastructure.

Acknowledgements

This work has been supported by Vinnova in the ITEA OPENPROD, MODRIO, and OPENCPS projects and in the Vinnova RTISIM project. Support from the Swedish Government has been received from the ELLIIT project. The OpenModelica development is supported by the Open Source Modelica Consortium. Many students, researchers, engineers have contributed to the OpenModelica system. There is not room here to mention all these people, but we gratefully acknowledge their contributions.

References

- Peter Aronsson. Automatic Parallelization of Equation-Based Simulation Programs. Linköping Studies in Science and Technology, Ph.D. Thesis No. 1022. June 14, 2006. URN: [urn:nbn:se:liu:diva-7446](http://nbn.se:liu:diva-7446)
- Adeel Asghar, Sonia Tariq, Mohsen Torabzadeh-Tari, Peter Fritzson, Adrian Pop, Martin Sjölund, Parham Vasaiely, and Wladimir Schamai. An Open Source Modelica Graphic Editor Integrated with Electronic Notebooks and Interactive Simulation. In *Proc. of the 8th International Modelica Conference 2011*, pp. 739–747. Modelica Association, March 2011. Linköping University, Sweden, 2010.
- Inderpreet Arora, Kannan Moudgalya, Sachitanand Malewar. A low cost, open source, single board heater system. In *Proc. 4th IEEE International Conference on E-Learning in Industrial Electronics (ICELIE)*, Nov 7-10, 2010. IEEE Xplore, DOI: 10.1109/ICELIE.2010.5669868
- Bernhard Bachmann, Lennart Ochel, Vitalij Ruge, Mahder Gebremedhin, Peter Fritzson, Vaheed Nezhadali, Lars Eriksson, Martin Sivertsson. Parallel Multiple-Shooting and Collocation Optimization with OpenModelica. In *Proceedings of the 9th International Modelica Conference (Modelica'2012)*, Munich, Germany, Sept.3-5, 2012
- Lutz Berger, Martin Sjölund, Bernhard Thiele. Code generation for STM32F4 boards with Modelica device drivers. In *Proc. of 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, Munich, Germany, Dec 1, 2017. Published by ACM Digital Library. doi:10.1145/3158191.3158204
- Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59: 65–98. 2017 doi: 10.1137/141000671.
- Willi Braun, Francesco Casella, and Bernhard Bachmann Solving Large-scale Modelica Models: New Approaches and Experimental Results using OpenModelica, In *Proc 12th Int. Modelica Conference*, May 15-17, 2017, Prague, Czech Republic, pp. 557-563, doi:10.3384/ecp17132557
- Dag Brück, Hilding Elmqvist, Sven-Erik Mattsson, and Hans Olsson. Dymola for Multi-Engineering Modeling and Simulation. In *Proceedings of the 2nd International*

- Modelica Conference*, Oberpfaffenhofen, Germany, Mar. 18–19, 2002
- Lena Buffoni and Peter Fritzson. Expressing Requirements in Modelica. In *Proceedings of the 55th Scandinavian Conference on Simulation and Modeling (SIMS'2014)*, available at www.scan-sims.org. Aalborg, Denmark, Oct 21–22, 2014.
- Lena Buffoni, Adrian Pop, and Alachew Mengist. Traceability and Impact Analysis in Requirement Verification. In *Proc. of 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, Munich, Germany, Dec 1, 2017. Published by ACM Digital Library. doi:10.1145/3158191.3158207
- Francesco Casella. Simulation of Large-Scale Models in Modelica: State of the Art and Future Perspectives. In *Proceedings of the 11th International Modelica Conference*, Sept 21–23 2015, Versailles, France, pp. 459–468, doi:10.3384/ecp15118459
- Alejandro Danós, Willi Braun, Peter Fritzson, Adrian Pop, Hugo Scolnik, and Rodrigo Castro. Towards an OpenModelica-based Sensitivity Analysis Platform Including Optimization-driven Strategies. In *Proc. of 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, Munich, Germany, Dec 1, 2017. Published by ACM Digital Library. doi:10.1145/3158191.3158206
- Hilding Elmqvist, Dag Bruck, and Martin Otter. *Dymola—User's Manual*. Dynasim AB, Research Park Ideon, SE-223 70, Lund, Sweden, 1996
- Dassault Systèmes. *Dymola. Systems Engineering Overview*. <https://www.3ds.com/products-services/catia/products/dymola/>. Accessed Sept. 3, 2018.
- 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.
- Rüdiger Franke, Marcus Walther, Niklas Worschech, Willi Braun, and Bernhard Bachmann. Model-based Control with FMI and a C++ Runtime for Modelica. In *Proceedings of the 11th International Modelica Conference*, Versailles, France, September 21–23, 2015. Published by LIU Electronic Press. doi:10.3384/ecp15118339
- Rüdiger Franke, Sven Erik Mattsson, Martin Otter, Karl Wernersson, Hans Olsson, Lennart Ochel, and Torsten Blochwitz. Discrete-time Models for Control Applications with FMI. In *Proceedings of the 12th International Modelica Conference*, Prague, Czech Republic, May 15–17, 2017. Published by LIU Electronic Press. doi:10.3384/ecp17132507
- Peter Fritzson, Lars Viklund, Dag Fritzson, and Johan Herber. High Level Mathematical Modeling and Programming in Scientific Computing, IEEE Software, pp 77–87, July 1995.
- 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>. An earlier version in *Proceedings of the 46th Conference on Simulation and Modelling of the Scandinavian Simulation Society (SIMS2005)*, Trondheim, Norway, October 13–14, 2005.
- Peter Fritzson. MathModelica - An Object Oriented Mathematical Modeling and Simulation Environment. *Mathematica Journal*, Vol 10, Issue 1. February. 2006.
- Peter Fritzson, Adrian Pop, David Broman, Peter Aronsson. Formal Semantics Based Translator Generation and Tool Development in Practice. In *Proc. of the 20th Australian Software Engineering Conference (ASWEC 2009)*, Gold Coast, Queensland, Australia, April 14 – 17, 2009.
- 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. February 2011. URL <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-68361>
- Peter Fritzson. *Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. 1250 pages. ISBN 9781-118-859124, Wiley IEEE Press, 2014.
- Peter Fritzson, Bernhard Bachmann, Kannan Moudgalya, Francesco Casella, Bernt Lie, Jiri Kofranek, Massimo Ceraolo, Christoph Nytsch Geusen, Luigi Vanfretti, (editors). *Introduction to Modelica with Examples in Modeling, Technology, and Applications*. Published by Linköping University Electronic Press, series "Linköping University Interdisciplinary Studies" with ISSN 1650-9625. On-line: <http://omwebbook.openmodelica.org/>. Accessed Sept 3, 2018.
- Dag Fritzson, Robert Braun, and Jan Hartford. Composite modelling in 3-D mechanics utilizing Transmission Line Modelling (TLM) and Functional Mock-up Interface (FMI) *Modeling, Identification and Control*, Vol. 39, No. 3, pp. 179–190, 2018.
- Peter Fritzson, Bernhard Bachmann, Kannan Moudgalya, Francesco Casella, Bernt Lie, Jiri Kofranek, Massimo Ceraolo, Christoph Nytsch Geusen, Luigi Vanfretti, (editors). *Introduction to Modelica with Examples in Modeling, Technology, and Applications* Published by Linköping University Electronic Press, series "Linköping University Interdisciplinary Studies" with ISSN 1650-9625. On-line: <http://omwebbook.openmodelica.org/>. Accessed Sept 3, 2018.
- Mahder Gebremedhin. *ParModelica: Extending the Algorithmic Subset of Modelica with Explicit Parallel Language Constructs for Multi-core Simulation*. Master Thesis, Department of Computer and Information Science, Linköping University, Oct. 2011. URN: [urn:nbn:se:liu:diva-71612](http://urn.kb.se/urn:nbn:se:liu:diva-71612)
- Mahder Gebremedhin and Peter Fritzson. Parallelizing Simulations with Runtime Profiling and Scheduling. In *Proc. of 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, Munich, Germany, 2017. Published by ACM.
- Julia org. *Julia Language Documentation*, Release 1.0. Accessed August 31, 2018, www.julialang.org
- Eva-Lena Lengquist-Sandelin, Susanna Monemar, Peter Fritzson, and Peter Bunus. DrModelica - An Interactive Tutoring Environment for Modelica. In *Proceedings of the*

- 3rd International Modelica Conference, Nov. 3-4, Linköping, Sweden, 2003
- Bernt Lie, Sudeep Bajracharya, Alachew Mengist, Lena Buffoni, Arunkumar Palanisamy, Martin Sjölund, Adeel Asghar, Adrian Pop, Peter Fritzson. API for Accessing OpenModelica Models From Python. In *Proceedings of the 9th Eurosim Congress on Modelling and Simulation*, EuroSim2016, Oulu, Finland, September 12-16, 2016. Published by IEEE, ISBN 978-1-5090-4119-0, pp. 707--713; <http://eurosim2016>.
- MathWorks. Matlab product overview. <https://www.mathworks.com/products/matlab.html> Accessed Sept 3, 2018.
- Nils Menager, Niklas Worschech, and Lars Mikelsons. A Toolchain for Rapid Control Prototyping using Rexroth Controllers and Open Source Software. In *Proceedings of the 10th International Modelica Conference (Modelica'2014)*, Lund, Sweden, March.10-12, 2014.
- Robert Milner, Mads Tofte, Robert Harper, and David MacQueen, *The Definition of Standard ML - Revised*. MIT Press. ISBN: 0-262-63181-4. Year 1997
- Modelica Association. *Modelica: A Unified Object-oriented Language for Physical Systems Modeling, Language Specification Version 3.4*. April 10, 2017. URL <http://www.modelica.org/>
- Modelica Association. SSP – MA Project for System Structure and Parameterization of Components for Virtual System Design. <https://www.modelica.org/projects> Accessed Sept 3, 2018.
- Kannan Moudgalya, Bhargava Nemmaru, Kaushik Datta, Priyam Nayak, Peter Fritzson, and Adrian Pop. Large Scale Training through Spoken Tutorials to Promote and use OpenModelica. In *Proceedings of the 12th International Modelica Conference (Modelica'2017)*, Prague, Czech Republic, May, 15-17, 2017.
- OCaml org. OCaml web site. <https://ocaml.org/> Accessed Sept 3, 2018.
- OSMC. *OpenModelica Users Guide*, latest version. <https://www.openmodelica.org/doc/OpenModelicaUsersGuide/latest/> Accessed September 3, 2018a.
- OSMC. OMSimulator 1.0 documentation. Chapter 6 in <https://www.openmodelica.org/doc/OpenModelicaUsersGuide/OpenModelicaUsersGuide-v1.12.0.pdf> Accessed September 3, 2018b.
- OSMC. OMSimulator 2.0 documentation: <https://openmodelica.org/doc/OMSimulator/html/> Accessed September 3, 2018c.
- Adrian Pop, Peter Fritzson, Andreas Remar, Elmir Jagudin, and David Akhvediani. OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging. In *Proceedings of the 5th International Modelica Conference (Modelica'2006)*, Vienna, Austria, Sept. 4-5, 2006.
- Adrian Pop and Peter Fritzson, MetaModelica: A Unified Equation-Based Semantical and Mathematical Modeling Language. In D. Lightfoot and C. Szyperski, editors, *Modular Programming Languages*, Vol. 4228 of Lecture Notes in Computer Science, pages 211{229. Springer Berlin / Heidelberg, 2006.DOI:10.1007/11860990 14.
- Adrian Pop. *Integrated Model-Driven Development Environments for Equation-Based Object-Oriented Languages*. Ph.D. Thesis. Linköping Studies in Science and Technology, Dissertation No. 1183, June 5, 2008.
- Adrian Pop and Peter Fritzson. MetaModelica: A Unified Equation-Based Semantical and Mathematical Modeling Language. In *Proceedings of Joint Modular Languages Conference 2006 (JMLC2006) LNCS 4228*, Springer Verlag. Jesus College, Oxford, England, Sept 13-15, 2006.
- Adrian Pop, Martin Sjölund, Adeel Asghar, Peter Fritzson, Francesco Casella. Integrated Debugging of Modelica Models. *Modeling, Identification, and Control*, Vol 35, No 2, pp. 93-107, DOI: <http://dx.doi.org/10.4173/mic.2014.2.3>, ISSN 1890-1328, Aug 2014.
- Python Software Foundation. Python Programming Language web page. <https://www.python.org/> Accessed Sept 3, 2018.
- Wladimir Schamai. *Model-Based Verification of Dynamic System Behavior against Requirements - Method, Language, and Tool*. Linköping Studies in Science and Technology, Dissertation No. 1547, Nov 12, 2013. DOI: [10.3384/diss.diva-98107](http://dx.doi.org/10.3384/diss.diva-98107)
- Wladimir Schamai, Lena Buffoni, Peter Fritzson. An Approach to Automated Model Composition Illustrated in the Context of Design Verification. *Modeling, Identification and Control*, Vol. 35, No. 2, pp. 79-91, ISSN 1890-1328, Aug. 2014
- Wladimir Schamai, Lena Buffoni, Nicolas Albarello, Pablo Fontes De Miranda, and Peter Fritzson. An Aeronautic Case Study for Requirement Formalization and Automated Model Composition in Modelica. In *Proceedings of the 11th International Modelica Conference (Modelica'2015)*, Paris, France, September, 21-23, 2015
- Martin Sjölund, Peter Fritzson, and Adrian Pop. Bootstrapping a Compiler for an Equation-Based Object-Oriented Language. DOI: 10.4173/mic.2014.1.1. *Modeling, Identification and Control*, Vol 35, No 1, pp 1-19, 2014.
- Martin Sjölund. *Tools and Methods for Analysis, Debugging, and Performance Improvement of Equation-Based Models*. Ph.D. Thesis. Linköping Studies in Science and Technology, Dissertation No. 1664, June 1, 2015.
- Bernhard Thiele, Thomas Beutlich, Volker Waurich, Martin Sjölund, and Tobias Bellmann. Towards a Standard-Conform, Platform-Generic and Feature-Rich Modelica Device Drivers Library. In *Proc. of the 12th Int. Modelica Conference*, Prague, Czech Republic, May 2017.
- Hubert Thieriot, Maroun Nemer, Mohsen Torabzadeh-Tari, Peter Fritzson, Rajiv Singh, and John John Kocherry. Towards Design Optimization with OpenModelica Emphasizing Parameter Optimization with Genetic Algorithms. In *Proceedings of the 8th International Modelica Conference (Modelica'2011)*, Dresden, Germany, March.20-22, 2011.
- Marcus Walther, Volker Waurich, Christian Schubert, and Ines Gubsch. Equation based parallelization of Modelica models. In *Proceedings of the 10th International Modelica Conference (Modelica'2014)*, Lund, Sweden, March.10-12, 2014.
- Volker Waurich and Jürgen Weber. Interactive FMU-Based Visualization for an Early Design Experience. In *Proc. of*

the 12th Int. Modelica Conference, Prague, Czech Republic, May 2017.

Stephen Wolfram. *The Mathematica Book*, 5th Ed. Wolfram Media, Inc, 2003.

Wolfram Research. Wolfram System Modeler Documentation and Overview. <http://www.wolfram.com/system-modeler/> Accessed September 3, 2018.

Andreas Wächter and Lorenz Biegler, On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming, *Mathematical Programming* 106 (2006) 25-57. Also: (Ipopt) <https://projects.coin-or.org/Ipopt>

Johan Åkesson. Optimica—An Extension of Modelica Supporting Dynamic Optimization. In *Proc. of 6th International Modelica Conference 2008*. Modelica Association, March 2008

Modelica on the Web

Tamas Kecskes¹ Patrik Meijer¹ Janos Sztipanovits¹ Peter Fritzson² Adrian Pop² Arunkumar Palanisamy²

¹Institute for Software Integrated Systems, Vanderbilt University, USA,

{tamas.kecskes,patrik.meijer,janos.sztipanovits}@vanderbilt.edu

²PELAB - Programming Environment Lab, Dept. of Computer and Information Science, Linköping University, SE-581 83 Linköping, Sweden, {peter.fritzson,adrian.pop,arunkumar.palanisamy}@liu.se

Abstract

Modelica has been around as a language from the late 1990's and since then a range of compilers and editors have emerged. Currently none of these environments provide a web-based user interface and follow the approach of requiring each end-user to install the application (typically together with a set of dependencies) on their local machine. This in itself may or may not be of major concern. Of more importance is their current lack of a seamless collaborative approach to modeling. This paper presents the first web-based collaborative graphical and textual modeling environment for Modelica based on WebGME and OpenModelica. Graphical composition of Modelica models from component libraries is supported via WebGME. Textual editing of the composite model is possible via OMWebBook.

Keywords: web-based modeling, collaborative modeling, metamodeling, WebGME, OpenModelica

1 Introduction

In the first part of this paper a web-based, online, collaborative Modelica (Fritzson and Engelson, 1998; Fritzson, 2014) modeling environment will be presented. It builds on WebGME (Generic Modeling Environment) and currently existing Modelica compilers, specifically OpenModelica and JModelica.org. This paper will demonstrate how WebGME, and to a certain degree metamodeling in general, can be used to rapidly create tailored modeling environments. The environment already includes key aspects such as collaboration, centralization of storage and distribution of computing resources.

This paper also gives an overview of OMWebBook and the potential integration points between these two web-based Modelica editors.

1.1 Terms and Acronyms used in this Paper

- Metamodel: defines the language and processes from which to form a model.
- DSML (Domain Specific Modeling Language): special-purpose languages designed to solve a particular range of problems.

- Strong relationship: child node is existence-dependent on the parent node.
- Meta-node: a node in a WebGME model hierarchy that is also part of the metamodel.
- Attribute: textual or numerical information as part of a node.
- Pointer: a named, directed, one-to-one relationship among nodes
- FCO: first class object that represents the atomic building block of a model in WebGME.
- ROOT: a container element that embeds all content of a project.

2 WebGME Background

WebGME is a web-based, visual modeling framework developed at Vanderbilt University (Maróti et al., 2014). The meta-programmable tool allows the creation of Domain Specific Modeling Language based modelers. It has a centralized, git-like model storage which allows multiple users to work on the same model simultaneously as well as creating separate branches that can be merged later. It also provides multiple extension points to allow fine-tuning of the user experience. The developer can define the visual appearance of different model elements or completely replace the entire model visualization. They can also implement their own model interpretation - like code generators or model verifiers - with the help of JavaScript based plugins.

3 Metamodeling

The WebGME framework provides a metamodeling paradigm resembling that of UML (Unified Modeling Language) (Lattmann et al., 2016). In contrast to traditional metamodeling frameworks, WebGME blurs the border between the metamodel and DSML models. Both are governed by the same datamodel that defines two strong relationships, *inheritance* and *containment*. These are single rooted trees over the same set of nodes with the inheritance root *FCO* (First Class Object) and the containment

root *ROOT*. The containment *ROOT* is not part of the inheritance tree nor the metamodel. The *FCO* is the prototypical *base* of all other nodes and defines the "name" attribute. Through prototypical inheritance, visualized as a red edge with a hollow arrow head in Figure 1, all nodes inherit the definition of the "name" attribute. Containment definitions are visualized as black edges with a filled diamond head. In Figure 1 the *FCO* is the *base* of *PortBase*, however a *ComponentBase* can contain a *PortBase*. All concepts in the metamodeling paradigm, except for inheritance, are definitions and should be viewed as *can have*s.

Pointer definitions in WebGME metamodeling environment are visualized as blue edges with an open arrow head as seen in Figure 2. *Connections* in WebGME are constructed using a pair of *pointers*, specifically *src* (source) and *dst* (destination). Nodes with these two pointer definitions are inferred to be *connections* and typically are visualized as edges when rendered in a DSML editor. The last metamodel concept used in this paper is the *Abstract* property. Nodes with this property cannot be instantiated in the model hierarchy. Even though a *Model* in Figure 1 can contain *ComponentBase*, the *ComponentBase* itself cannot be instantiated; however, the *Modelica.Mechanics.Translational.Components.Mass* from Figure 3 can be instantiated. For a full overview of the metamodeling concepts of WebGME refer to (Meijer and Mavridou, 2018) or (Maróti et al., 2014).

3.1 The Modelica Domain

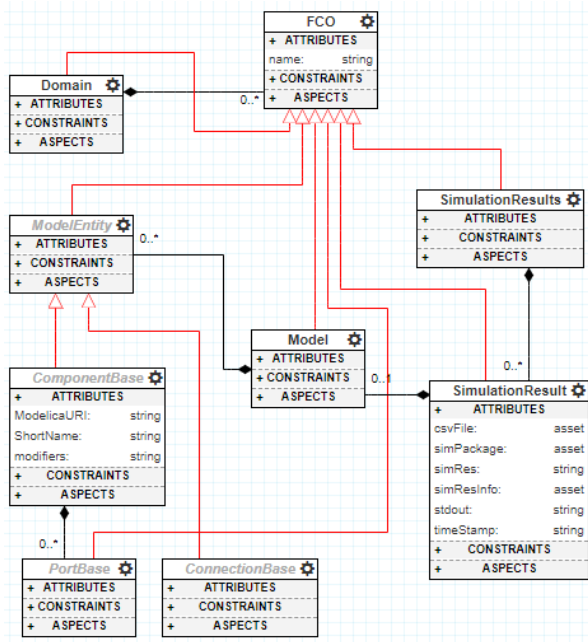


Figure 1. The base concepts for the Modelica Domain in WebGME

Modelica is at its core a declarative equation-based language extended with a rich type system, object-oriented features, functions, graphical annotations etc. (Fritzson, 2014). In close relation with the language is its standard

library, the MSL (*Modelica Standard Library*). MSL is maintained by *The Modelica Association* and currently offers over 1300 components, many of which are ready-to-use, parameterized building blocks for modeling of complete dynamic systems. The target of the DSML and modeling environment presented in this paper is to offer a graphical editor building such systems. The approach taken here is to extract the interfaces and parameters from the MSL components and store this information as prototypes in WebGME. Before constructing any of these prototypes or meta-nodes, a metamodel with the base concepts was created. In Figure 1 the central concept is the *Model*, which can contain *ComponentBases* and *ConnectionBases*. In the finished editor the *Model* will act as the "drawing canvas" where dynamic systems of MSL components will be composed. *ModelicaURI* is an important attribute defined at the *ComponentBase*. In the derived MSL components in WebGME the *ModelicaURI* gets populated with the unique path to the associated MSL component.

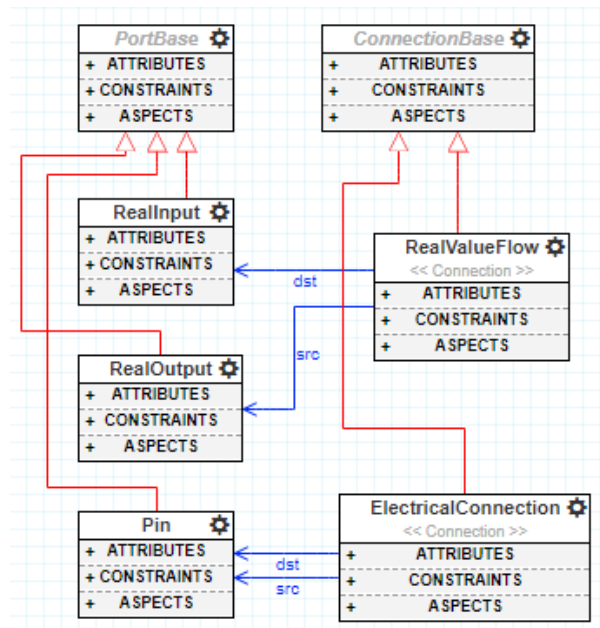


Figure 2. Metamodel for the Modelica connectors

On the left-hand-side of Figure 3 a sample of MSL components in WebGME are presented from the meta-view. To avoid name collisions, the *ModelicaURI* is also used as the name attribute of the meta-nodes. On the right-hand-side of Figure 3 the same components are displayed from the containment view. In contrast to the meta-view, this view shows what actual characteristics these prototypes *do* have, rather than what they *can* have. Those familiar with MSL probably recognize the flange connectors appearing as semicircles along the borders of the boxes. These connectors are not technically part of the metamodel. When prototypes are instantiated by the user on the canvas (mid section in Figure 5) the instances inherit all properties of their prototype. This not only includes the default values of the attributes, but also the

meta-definitions and the contained children nodes.

The MSL components are imported using a script that extracts the connectors and parameters from the Modelica code using the OpenModelica parser. WebGME nodes are created from the extracted data and organized in *Domain*-folders. The base metamodel contains predefined *Ports* and *Connections* that map to connectors of MSL for the purpose of capturing compatibility between different Modelica connectors. In Figure 2 a sample of these are presented. In the case of acausal connections, where direction does not matter, the source and destination are interchangeable. For directed value flows however, the connections are forced to be made from output to input. Constraints like these are captured by the WebGME API during modeling and although not necessary for a Modelica domain - a single port and connection concept would suffice - it allows for a more guided user-experience.

Up to this point the portion of the metamodel needed for governing the composition of systems of Modelica components from the Modelica Standard Library has been explained. The WebGME-DSS framework supports generation and simulation of Modelica code, see section 4.3, and in order to provide a closer link between models and simulation results the concept of *SimulationResult* is introduced, see Figure 1. Instances of these are created whenever a Model is invoked to be simulated. As the simulation progresses the different attributes are populated with the inputs and outputs to and from the Modelica compiler and runtime environment. In order to not overflow the models with large amount of data the *asset* attribute of WebGME is utilized. The asset attribute only stores a lightweight SHA-256 hash in the model acting as a key to the underlying artifact (typically stored at the file-system of the server). To maintain a mapping between the results and models, the *SimulationResult* node is populated with a copy of the Modelica model enabling the user-interface to map the plotted variables back to the graphical model.

4 The User Interface

In the current section we will introduce the WebGME Dynamic Systems Studio, that has the goal of introducing Modelica on a beginner level by allowing visual composition of dynamic systems composed of components from the Modelica Standard Library.

4.1 Project Organization

The landing page - shown on Figure 4 - has two main functionalities. The user can create a new project either by selecting a specific domain from MSL as the basis of the project or by choosing the *Hybrid Domain* that allows selection of multiple libraries. These selections are not final, and only instructs the system to seed the project with the necessary information and can be changed any time during model editing.

The page also lists all the projects that the user has access to. The items not only show the name of the project, but highlight the applied libraries, giving extra informa-

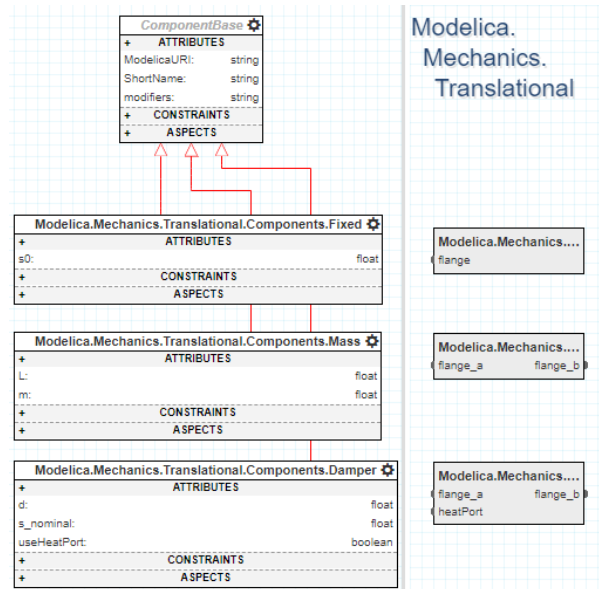


Figure 3. MSL models from the meta- and containment-perspective.

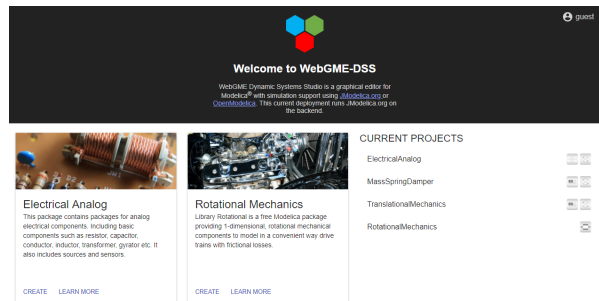


Figure 4. The landing page listing the user's current projects.

tion to the user. To modify access rights of projects, create organizations, view information about other users on the same deployment, etc. a link to the default WebGME profile page is available in the top-wright corner.

4.2 Modeling

The main page of the tool is the editor itself (Figure 5 and 6). It has two views - selectable on the bottom center of the page - the *Modeling* where the user can compose and initiate simulations of the system, and the *Results* where the progress of simulations and finally the time traces of the simulated variables can be plotted. The toolbar at the top provides zooming function to help in the navigation on larger, more complex systems. It also have a saving button to allow creating notes for the current version of the model.

The left sidebar hosts a part-browser where the elements of the used libraries are listed. These elements can be instantiated and put into the system by a simple drag and drop onto the main canvas. In addition to the available components, the sidebar also implements several buttons for important features. The first in the list is a check function, that verifies if a syntactically correct Modelica

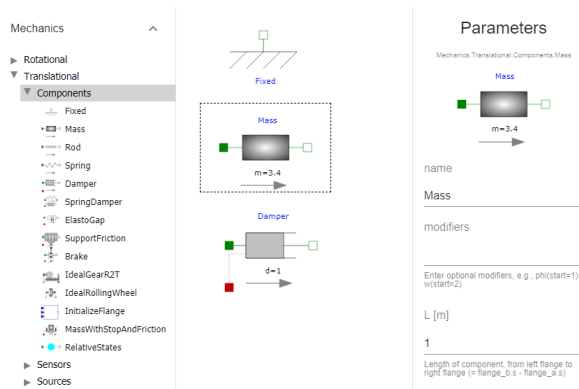


Figure 5. From left to right; the part-browser, canvas and attribute/parameter editor.

code can be generated from the model. If something is wrong - like two elements have the same name - the result dialog lists the error providing a link, that will select the faulty node. The second button initiates a simulation. First, the user provides some basic parameters regarding the simulation, then, depending on how the server is configured, the collection of the generated necessary artifacts are either downloaded or the simulation gets started on the server. The third button brings up a multi-selection list of the available libraries. The final button shows the history dialog. The default view of the dialog lists the versions that were marked by the user - with the help of the save button - plus the current state of the project. As the WebGME creates micro-commits and stores even the smallest change, the user can switch to a *detailed* view, where all the created commits are visible. For every entry of the list, the user can reset the work to that given state or the difference between that and the current version can be visualized.

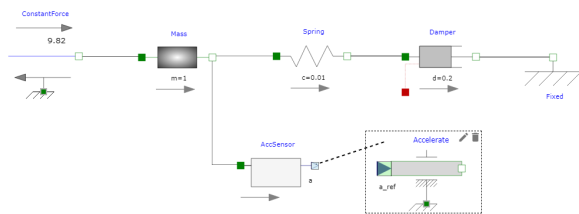


Figure 6. The port compatibility defined in the metamodel reflected in the modeling view.

With components on the main canvas, the user can edit the parameters by double-clicking on it or selecting and clicking the edit button. Hovering over the interfaces of the nodes, the user can initiate the creation of a connection. In connect mode, a dashed line from the source interface to the mouse pointer notifies the user about the ongoing task. Whenever the user reaches a valid target port-node, those interfaces will be highlighted guiding the user. If clicked on a correct endpoint, the connection is made while leaving the main canvas or clicking anywhere else cancels the operation. Each element visualizes its pa-

rameters according to its specification to give the user the most information upfront.

4.3 Simulation and Results

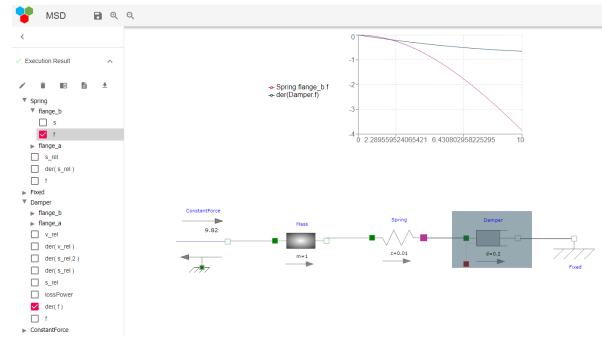


Figure 7. The plotted variables are highlighted in the originating model.

The *result* view of the tool provides the same areas, but with modified functionalities. The side-bar contains the list of the simulations. The items are highlighted according to their state - whether they are running or completed. Once the simulation is finished, the structured list of variables becomes available, and by selecting the interesting ones, the user will initiate a plot visualization. The plot is visualized in the top half of the main canvas, while the bottom half shows the model. The model version shown is a copy of the one at the time of the start of the simulation - and this view is read-only - and the portions that own the plotted variables are highlighted with matching colors to inform the user about the content of the chart. To allow the user to analyze the result in depth, the chart can be detached from the screen - with the button at the top right corner - which changes the upper portion of the main canvas allowing the creation of multiple charts and individual control of their content. Once the user is done with the detached graphs they can be closed with a single button click. If the server run into issues during execution or the simulation is not ready, the top half of the main canvas will show the console output of the execution of the simulation engine.

5 System Architecture

There are a number of interacting components of the system that provides the user the WebGME-DSS interface described in the previous section. In the center of the system lies the WebGME server. Multiple instances can serve a single deployment which allows greater capacities in terms of connected users and a way of horizontal scaling. To enable multiple servers to work together, a Redis memory database needs to be added to the system. Through a publish/subscribe service, the Redis (Redis) provides fast and efficient communication between servers that allow notification to users who might be working on the same project but connected to different servers. Also an HTTPS reverse proxy - like Nginx (NGINX) - becomes necessary

<https://webgme-dss.isis.vanderbilt.edu>

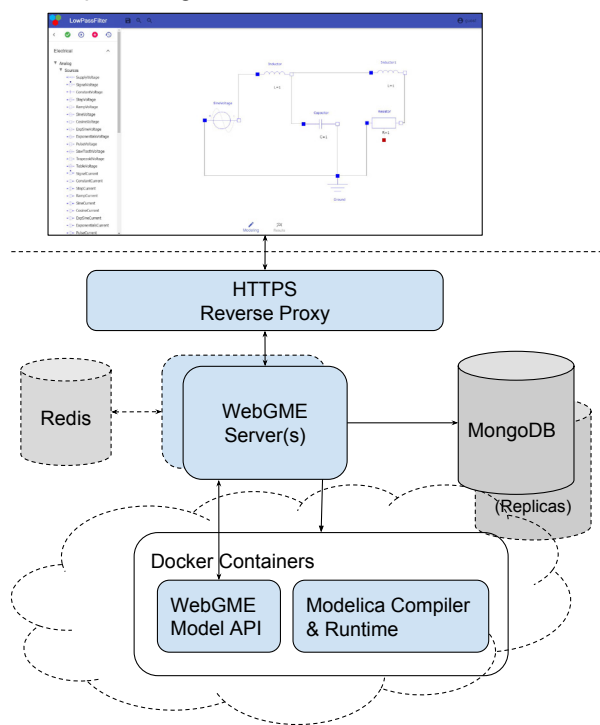


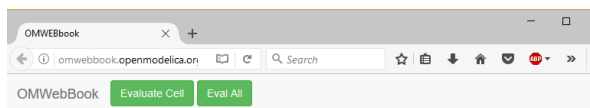
Figure 8. Overview of the different applications and services making up the framework.

to route the requests based on their session id, enabling users to communicate with a single server. This proxy also enhances the security of communication so its use should be considered in a single server configuration as well. MongoDB (MongoDB) is responsible for storing all project related data and multiple instances - so called shards - can be leveraged for bigger scalability. The final actors of the system are the Modelica simulators. The server connects to them and runs the simulation, gather the results. They are usually packaged in a Docker (Docker) container for easy deployment and minimal configuration. when it comes to vertical scaling, these components can live in a single computer or spread throughout multiple ones.

6 OMWebBook

There is currently a strong trend in integrating documents and computation. This is most visible in web technology where computational facilities are increasingly being embedded within web pages. However, facilities for user programming and mathematical modeling are still largely absent in web pages. Mathematica (Wolfram, 2003) pioneered the idea and implementation of active interactive electronic notebooks. The Mathematica notebook facility is an interactive WYSIWYG (What-You-See-Is-What-You-Get) realization of Literate Programming, a form of programming where programs are integrated with documentation in the same document. However, the original implementation of Literate Programming was a non-

interactive system integrated with the document processing system LaTeX.



First Basic Class

1 HelloWorld

The program contains a declaration of a class called HelloWorld with two fields and one equation. The first field is the variable x which is initialized to a start value 1 at the time when the simulation starts. The second field is the variable a , which is a constant that is initialized to 1 at the beginning of the simulation. Such a constant is prefixed by the keyword parameter in order to indicate that it is constant during simulation but is a model parameter that can be changed between simulations.

The Modelica program solves a trivial differential equation: $x' = -a * x$. The variable x is a state variable that can change value over time. The x' is the time derivative of x .

```
1 class HelloWorld
2   Real x(start = 1, fixed=true);
3   parameter Real a = 1;
4   equation
5     der(x) = - a * x;
6 end HelloWorld;
```

2 Simulation of HelloWorld

```
1 simulate( HelloWorld, startTime=0, stopTime=4 )
2
```

Plot the results.

```
1 plot( x )
2
```

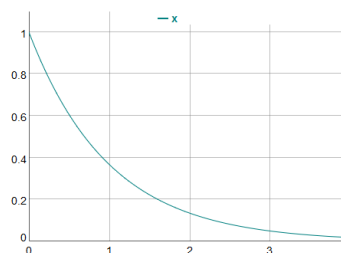


Figure 9. OMWebBook with editable models, simulations, and plots.

Traditional documents, e.g. books and reports, essentially always have a hierarchical structure. They are divided into sections, subsections, paragraphs, etc. Both the document itself and its sections usually have headings as labels for easier navigation. This kind of structure is also reflected in electronic notebooks. Every notebook corresponds to one document and contains a tree structure of cells. A cell can have different kinds of contents, and can even contain other cells. The notebook hierarchy of cells thus reflects the hierarchy of sections and subsections in a traditional document. The OMNotebook notebook facility in OpenModelica supports several kinds of contents, for example cells with executable Modelica model classes, commands, documentation, pictures, and 2D graphs. However, OMNotebook is an off-line tool, part of the OpenModelica tool suite, that has to be installed before use.

Therefore, we have developed OMWebBook (Fritzson, 2017), (Figure 9) which is an on-line interactive web-based electronic book (<http://omwebbook.openmodelica.org/>). This is similar to OMNotebook, but textual model editing and simulation is per-

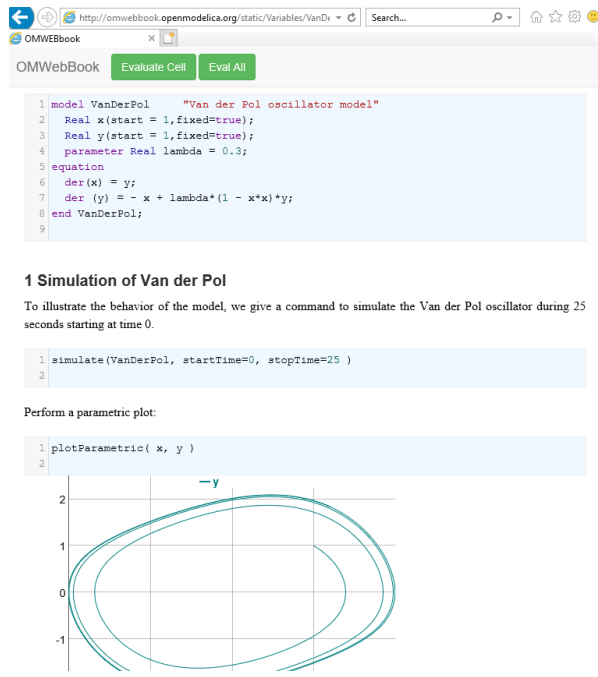


Figure 10. The VanDerPol sub-document showing a cell with a Modelica model, simulation commands, and plot results.

formed in a web-browser. Simulation is performed by a dedicated simulation server. Thus, the user need not install OpenModelica on a computer in order to edit models and run simulations. Editing and simulation can even be done from smartphones or tablets. Figure 9 shows part of the OMWebBook introductory Modelica teaching material called DrModelica (Lengquist-Sandelin et al.), starting with the simplest possible HelloWorld model that the student can edit, simulate and plot in the web browser.

Figure 10 shows the model, documentation, simulation and plotting of the VanderPol model.

However, so far OMWebBook has had one big drawback - it has lacked support for web-based graphic editing of Modelica models. This can be seen in the chapter about simplified modeling of electric and hybrid vehicles, starting with the simple electric vehicle graphical model depicted in Figure 11.

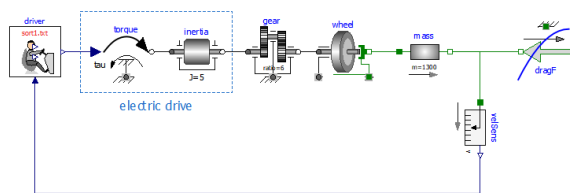


Figure 11. Connection diagram of a first, very simple, electric vehicle model.

The reader was previously instructed to download and install OpenModelica in order to edit and simulate this graphic model. However, this is no longer necessary with the combination of the WebGME web-based graphic

model editor and the OMWebBook text editing and simulation capabilities. Now, also graphical models can be embedded in an on-line electronic book and edited and simulated on the web.

7 Future Work

This proof of concept implementation proved to be an efficient integration platform that can bring Modelica closer to end-users and allow them to focus solely on the system they try to describe. By integrating analysis tools we plan to provide a broader spectrum of functions to the users. Also, by better integrating the OMWebBook code-editor into the WebGME-DSS the user will be able to describe more detailed behavior for the components of their systems. The code editing also supports syntax highlighting as well as checks for compilation errors. Future work will address the enlargement of available components. This can be done in multiple ways. More MSL elements will be curated into the system and a library importer will allow developers to include their custom Modelica libraries. Finally, a special visualization will provide a platform for the user to create new components by using inheritance and a visual description language.

References

- Docker. Docker - Build, Ship, and Run Any App, Anywhere. <https://www.docker.com/>. Cited 2018 May 11.
- Peter Fritzson. *Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley IEEE Press, 2014. ISBN 9781-118-859124.
- Peter Fritzson. Introduction to Modelica with Examples in Modeling, Technology, and Applications using OMWebBook <http://omwebbook.openmodelica.org/>. Technical report, Linköping University, PELAB - Programming Environment Laboratory, 2017.
- Peter Fritzson and Vadim Engelson. Modelica — a unified object-oriented language for system modeling and simulation. In Eric Jul, editor, *ECOOP'98 — Object-Oriented Programming*, pages 67–90, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN 978-3-540-69064-1.
- Zsolt Lattmann, Tamás Kecskés, Patrik Meijer, Gábor Karsai, Péter Völgyesi, and Ákos Lédeczi. Abstractions for modeling complex systems. In *International Symposium on Leveraging Applications of Formal Methods*, pages 68–79. Springer, 2016.
- Eva-Lena Lengquist-Sandelin, Susanna Monemar, Peter Fritzson, and Peter Bunus. Drmodelica - a web-based teaching environment for modelica.
- Miklós Maróti, Tamás Kecskés, Róbert Kereskényi, Brian Broll, Péter Völgyesi, László Jurácz, Tihamer Levendovszky, and Ákos Lédeczi. Next generation (meta) modeling: Web-and cloud-based collaborative tool infrastructure. *MPM@ MoD-ELS*, 1237:41–60, 2014.
- Patrik Meijer and Anastasia Mavridou. How to build a design studio with webgme. 05/09/2018 2018. ISSN ISIS-18-101.

MongoDB. MongoDB: MongoDB for GIANT Ideas. <https://www.mongodb.com/>. Cited 2018 May 11.

NGINX. NGINX: High Performance Load Balancer, Web Server, Reverse Proxy. <https://www.nginx.com/>. Cited 2018 May 11.

Redis. Redis. <https://redis.io/>. Cited 2018 May 11.

Stephen Wolfram. *The Mathematica Book*. 5th Ed. Wolfram Media Inc, 2003. ISBN 1579550223.

A Modelica Library for Thin-Layer Drying of Agricultural Products

Augusto Souza¹ Brian Steward¹ Carl Bern¹

¹Agricultural and Biosystems Engineering, Iowa State University, USA,
{acsouza,bsteward,cjbern}@iastate.edu

Abstract

Grain drying is highly influenced by environmental and technical factors. Thus, it is essential to track the psychrometric properties of the drying air, besides other grain characteristics, for successful control of this operation. Mathematical modeling of a drying process can be complicated and non-trivial when considering all the involved factors. Based on theoretical differential equations, this study calculates different aspects of grains during their drying process. Modelica and Dymola were used to model blocks of thin-layers of corn, barley, and soybean. The modeled blocks could be used to reproduce a simulation of a grain drying process and keep track of the products moisture content and temperature, besides other psychrometric properties of the air. The developed model has the potential to be used to either to compare to a real grain drying process or as a teaching instrument for grain handling.

Keywords: agriculture, grain drying, Modelica, corn, soybean, barley

1 Introduction

Moisture content (MC) is an important factor when managing stored grains in agriculture. The amount of water in a agricultural product can determine its value, overall quality, and commercialization. Excess moisture can result in spoilage and insect proliferation. Under improper conditions, stored grain can be a vector of harmful fungi that produce carcinogenic aflatoxins (Christensen and Kaufmann, 1965; Lacey, 1989). Meanwhile, overdried grains result in reduced mass when sold and in wasted drying energy.

Grain drying is a commonly used process to remove water from grains. For commercialization purposes, the USDA assigns grades for grains depending on the quality of the product (USDA, 2013). The amount of damage caused by heat is one of the factors that determine the final grade. Additionally, careless grain drying can decrease grain's overall quality. Indeed, Wall et al. (1975) showed that corn can lose protein content during drying for temperatures equal or higher than 160 °C. Thus, a negligent control of the drying process can lead to a decrease in grain quality and, consequently, commercial value.

Grain drying is profoundly influenced by different en-

vironmental and technical factors. For instance, the psychrometric condition of the heated air will determine the course of the drying process. Thus, it is essential to monitor these thermodynamic and physical properties for successful control of the operation. Additionally, the dehydration of biological material is, likewise, influenced by its chemical composition, actual moisture content, initial conditions, and other aspects. Depending on the type of dryer, the moisture content of the product will be time and space dependent. Thus, it is challenging to predict the exact MC value within this non-linear process (Brooker et al., 1992). Mathematical modeling of a drying process can be non-trivial when considering all the factors involved. Modelica has the advantage of organizing digitally-described components in libraries that can be exported to distinct models and used in different applications. For instance, an encapsulation of grain properties into objects could be used to simulate different drying situations without the necessity of coding and programming.

An early report documented the design of an analog computer for deep-bed drying grain simulation (Baughman et al., 1971). The authors developed equations to calculate moisture content as a function of time and location for dry shelled corn. With this work, an analog computer was able to accurately estimate MC for the grain. There are more recent examples of modeling and simulating grain drying using different tools and techniques. Khatchatourian et al. (2013) developed software to solve a mathematical model of a cross flow dryer for soybeans. The model was validated using a test stand developed by the authors. The developed model showed a good fit with the validation model, including the energy saved using the cross flowing drying method. However, it would be interesting to see more flexibility in the types of grain used and their drying characteristics. More advanced methods have been used to study the drying process. Azmir et al. (2018) used a combination of computational fluid dynamics (CFD) and discrete element method (DEM) techniques to model and simulate grain drying in a fluidised bed for corn, where the simulation data were consistent with the experimental data. Additionally, Jia et al. (2002) developed computer simulation software to calculate drying for a single grain kernel. According to the study, investigating a single kernel in a bulk of grains can indicate the overall quality of the mass. The authors studied the effects of sev-

eral variables in different grains using MATLAB 6.0 ®. Although a graphical user interface (GUI) was developed to facilitate use, this software requires a paid license prior to use. It would be difficult to have access to such a tool even for those that can afford it. Modelica has the advantage of being used with open-source software, and with a relative ease of use. Even though it has been used to model different biological or thermal processes (Jain et al., 2017; Aourousseau et al., 2015), few examples exist of its use for agricultural purposes.

Modelica can be used in simulation platforms to design models of dynamic systems. Simulation is used to analyze models and compare the results with what is expected in a real-life situation. Dymola is one platform where Modelica is used to simulate physical models through its libraries. Dymola has the flexibility to change how a system could work by modifying parameters inside each component, how they are connected, and other different conditions.

The objective of this work was to develop a grain drying process model for barley, corn, and soybean by taking into consideration the most relevant aspects that influence the process. Even though soybean is an oilseed, during the course of this paper it is referred as a grain. The Modelica language was used with the Dymola (Version 2017, Dassault Systèmes) environment to create models and packages related to this content. Simulation results were compared to experimental data and the error between these two were calculated. The models could be an agricultural tool for grain monitoring, as well as implemented for didactic reasons for students learning about the drying process.

2 Methods and System Modeling

2.1 Theoretical Modeling of a Thin Layer Drying

For modeling purposes, grain drying is a thermodynamic process of simultaneous heat and moisture transfer. During this process, air temperature changes as it crosses the layer of grains, as can be observed from Figure 1. Latent heat in the air is transferred to the grain, and at the same time, free water is removed from inside the grains. For this model, the air would flow from the bottom and up through all the grain layers until the exiting at the top.

The humidity ratio is defined as the mass of water vapor per unit mass of dry air. This psychrometric air property is dependent on air temperature and relative humidity. Therefore, during drying, the air moving across the grain layer will remove the moisture, increase the humidity ratio, and, consequently, decrease the capacity of the air to remove water from the grain in the next layer.

The equilibrium moisture content (M_e) of the grain is also dependent on the psychrometric properties of the air. M_e is achieved when the grain internal vapor pressure is equal to the vapor pressure of the environment (Brooker et al., 1992). That is, if the vapor pressure of the environment is less than the vapor pressure inside the prod-

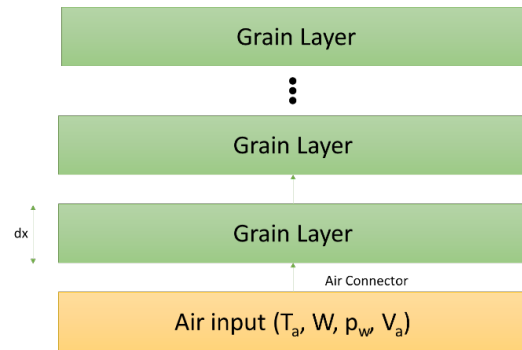


Figure 1. Schematic of the elemental stationary thin layer (thickness = dx) connected with other layers

uct, the material will lose (desorb) moisture. When the air is heated, this vapor pressure decreases; however, as the air removes the moisture from the grain, the pressure increases and therefore reduces the drying ratio of the operation.

Differential equations, based on laws of heat and mass transfer, and psychrometric equations, were used according to different sources (Brooker et al., 1992; Pabis et al., 1989). Assumptions are made for a thin layer modeling:

1. The volume shrinkage of the kernels was negligible for the calculations.
2. The temperature gradients for an individual layer were constant.
3. The model did not take into consideration the conduction between kernels.
4. The dryer walls were considered adiabatic and with negligible heat capacity.
5. Empirical and theoretical equations were used for parameter calculation. All these equations were considered accurate.

Four major dynamic variables were calculated in this model as functions of time:

- M - Moisture content of the grain (decimal, dry basis)
- T_p - temperature of the grain ($^{\circ}C$)
- W - humidity ratio (kg of water vapor per unit kg of dry air)
- T_a - temperature of the air ($^{\circ}C$)

Equations 1 from 4 were used to analyze the grain drying behavior during time of a thin layer.

$$\frac{dM}{dt} = -k \times (M - M_e) \quad (1)$$

$$\frac{dT_a}{dt} = -\frac{h'a}{G_a c_a + G_a c_v W} \times (T_a - T_p) \quad (2)$$

$$\frac{dT_p}{dt} = \frac{h'a \times (T_a - T_p) + [h_{fg} + c_v(T_a - T_p)] \times (G_a \frac{dW}{dx})}{\rho_p c_p + \rho_p c_w M} \quad (3)$$

$$\frac{dW}{dx} = -\frac{\rho_p}{G_a} \times \frac{dM}{dt} \quad (4)$$

$$MR = \frac{M - M_e}{M_0 - M_e} \quad (5)$$

The model used to calculate the equilibrium moisture content (M_e in dry basis or d.b.) was the modified Henderson equation solved for M_e (Brooker et al., 1992);

$$1 - \frac{P_v}{P_{vs}} = e^{-K_{M_e}(T_a + C_{M_e})(100 \times M_e)^{N_{M_e}}} \quad (6)$$

Where,

P_v - vapor pressure of water (Pa)

P_{vs} - saturated vapor pressure of water (Pa)

$K_{M_e}, C_{M_e}, N_{M_e}$ - constants associated with equation 6, as presented in Table 1

Where,

k - drying constant (h^{-1})

M_e - equilibrium moisture content (decimals, wet basis w.b.)

h' - convective heat transfer coefficient ($J.m^{-2} \circ C^{-1} h^{-1}$)

a - particle surface area per unit bed volume ($m^2.m^{-3}$)

G_a - air flow rate ($kg.h^{-1}m^{-2}$)

c_a - specific heat of dry air ($J.kg^{-1} \circ C^{-1}$)

c_v - specific heat of water vapor ($J.kg^{-1} \circ C^{-1}$)

c_p - specific heat of dry grain kernels ($J.kg^{-1} \circ C^{-1}$)

h_{fg} - heat of evaporation ($J.kg^{-1}$)

dx - layer thickness (m)

ρ_p - density of the material ($kg.m^{-3}$)

These equations are highly dependent on temperature and moisture differences. The rate at which moisture changes in the mass of grain will depend on its equilibrium moisture content (M_e), which, as the name suggests, will be a point where the grain will be neither losing or absorbing water. This equilibrium is reached depending on the air temperature, relative humidity, and other air temperatures, which are also variables during this dynamic process. Thus, this set of equations physically influence and are dependent of each other.

For this case, the layer of grain was considered thin, in such a way that, the gradient of a variable in a single layer is linear. In other words, the solution of a variable derivative related to the thickness (for example, dW/dx) would simply be the difference of this variable value in the beginning and at the end of the layer divided by the thickness of the same ($\Delta W/dx$). With these four essential equations and psychrometric functions, a drying model was designed using the Modelica software.

Moisture Ratio (MR) is calculated using equation 5 and can be interpreted as the portion of water that can still be removed from the grain, using the equilibrium moisture content as a reference. That way, the MR of different grains can be compared independent of the initial moisture content.

Table 1. Constants associated to the Equilibrium Moisture Content Equation

| Constant | Barley | Corn | Soybean |
|-----------|-----------------------|-----------------------|------------------------|
| K_{M_e} | 2.29×10^{-5} | 8.65×10^{-5} | 30.53×10^{-5} |
| N_{M_e} | 2.0123 | 1.8634 | 1.2164 |
| C_{M_e} | 195.267 | 49.810 | 134.136 |

To model the drying process, thermal and physical properties for each grain had to be calculated and this, also, had an influence in the procedure. These properties were equivalent particle radius, specific heat of the product, latent heat of vaporization, grain density, and the product drying constant. Each of these variables had associated constants that were used in empirical models to be calculated. It is beyond of the scope of this paper to list the value of all of them, but they were retrieved from different sources for all the studied grains (Bortolaia et al., 2010; Brooker et al., 1992; Bruce, 1985; Otten and Samaan, 1980).

2.2 Modelica Models

The models were developed using the Modelica language. Two main packages were created, *Dryer* and *Grain*. The Dryer library contained all the models and classes related to the dryer while the Grain library had the objects modeled associated to the grains and their features.

For the *Dryer* library, an *AirCon* connector was created to pass the values for air temperature (T_a), pressure of water vapor (p_w), air humidity ratio (W), and air velocity (Va), as noted in the code:

```
connector AirCon
import SI = Modelica.SIunits;
SI.Temperature Ta(displayUnit="Kelvin") "
  Air temperature (K)";
pw "partial pressure of water vapor in
  air";
```

```

SI.MassFraction W(min=0) "air humidity
  ratio";
SI.Velocity Va(start=10, displayUnit="10"
) "Air velocity";
end AirCon;

```

These are features that were used in all the models that required air properties. Also, this connector was used to calculate the psychrometric properties of the air at any point in the drying process. For the Grain library, this same unidirectional-flow *AirCon* was used for the grains' layers object-model.

Inside the Dryer package was located the *Dryer_Input* and *AirExit* models, besides the *AirCon* connector. *Dryer_Input* was used to describe the initial conditions of the dryer. It was possible to input the initial air temperature ($^{\circ}\text{C}$), the drying temperature ($^{\circ}\text{C}$), initial relative humidity (%), dryer diameter (meters), and air velocity (m/sec).

For each grain, a Layer model was designed inside the *Grain* Package, which contained the variables and constants associated with the grain. Each Layer had two *AirCon* connectors for the air input and output. The connector at the exit of the layer could be linked to the next layer. Thus, the air that leaves the first layer would be used to dry the second one. The initial thickness (m) and MC (decimal, w.b.) of the grain layer could be logged in this model.

A dryer operation simulation contained a stack of thin-layers, modeled in Dymola, attached to the air input, as observed in Figure 2. The layers were connected in such manner that the air used in the first layer would exit and enter the second layer with different psychrometric properties; thus, having the properties after the air removes moisture from the initial layer. At the end, the final psychrometric properties were calculated in the developed *Output* class. This class represents the exit of the dryer and it could be used to observe the psychrometric properties of the exiting air.

The time unit used for the simulation was *seconds*, even though the results are reported in *hours*. This distinction is important to emphasize because the simulation steps are based on this unit and conversions should be made convenient for the involved variables. The International System

of Units (SI) was used for all other units using the Modelica Standard Library.

Different scenarios were simulated to test the model. Three thin layers of the studied grains were dried under the same conditions and duration. The initial moisture content was 35% (w.b.) and the layer was 0.25 meters thick. The grain temperature was originally at 25°C , and the drying air temperature was 70°C for this simulation. The relative humidity of the air, before entering the Dryer, was set as 50%. The drying duration was two hours and the simulation used the Dassl algorithm (tolerance = 0.0001) to find the solution for this model. Additionally, fifty thin layers, with a thickness of 0.25 meters each, were stacked, representing the drying of a deep bed column of 12.5 meters of grain. The simulation had a duration of 10 hours. The outside air was set to a temperature of 30°C and relative humidity of 50%. Also, the effect of three different temperatures were investigated for thin layers of corn. Three layers of corn, with an initial product temperature of 15°C , were modeled at three different temperatures (25°C , 50°C , and 90°C) during a drying period of 3 hours.

To validate this model, the simulation results were compared to different experimental values using similar characteristics of the designed model results (Table 2). For instance, all the works consulted used thin layers of grains in static beds at different temperatures (Markowski et al., 2010; Li and Morey, 1984; Freire et al., 2005).

3 Modeling Outcome and Simulation Results

The two main components modeled were the grain layers models and the air input ("*Grain*"*Layer* and *Dryer_Input*). Some of the parameters for both components would take default values if not specified by the user. For the *Dryer_Input* component, the inputs needed were Outside Air Temperature (T_0), desirable Drying Temperature (T_a), Relative Humidity (RH), Bin Diameter (D), and Air Flow Velocity (V_a). All these were parameters of the *Dryer_Input* component. The *CornLayer* component needed three inputs: Thickness of the grain layer (dx), Initial MC (M_0), and Initial Temperature of the Product (T_p).

Table 2. Experimental data collected for comparison

| Parameters | Barley (Markowski et al., 2010) | Corn (Li and Morey, 1984) | Soybean (Freire et al., 2005) |
|--|---------------------------------|---------------------------|-------------------------------|
| Initial Moisture Content (% , w.b.) | 17.5 | 26 | Results in Moisture Ratio |
| Layer Thickness (mm) | 333.0 ± 5.0 | 5.91 | 27.0 |
| Air velocity (m/s) | 30.1 ± 0.1 | 0.3 | 1.75 |
| Initial Air Temperature ($^{\circ}\text{C}$) | 33, 41, 48, and 56 ± 2 | 27, 49, 71, 93, and 116 | 31.5, 45, and 58.5 |
| Initial Product Temperature ($^{\circ}\text{C}$) | Around 10 | Room temperature | Close to air temperature |
| Drying duration (hours) | 3 | 10 | 6.666 |
| Initial RH (%) | ± 35 | - | - |

The first two were parameters of the model, while the last one was considered as a variable since it changes with time as it exchanges heat with the air.

3.1 Thin-layer models

The drying of thin layers was simulated for each grain studied in parallel (Figure 3(a)). Although Figure 3 shows the source of hot air from the same dryer, there was no separation in the flow of air in the simulation; they all had the same input air properties from the *Dryer_Input* block.

The change in the Moisture Ratio (MR) was observed for the three grains (Figure 3(b)). At the end of the simulation, the final MR value for corn was 0.319, 0.300 for soybean, and 0.074 for barley, thus, the latter had a faster drying rate compared to corn and soybean at 70 °C. This difference is expected since the grains have different properties based on their chemical compositions and physical characteristics that influence the exchange of water with the air. However, for this example, two hours was not enough to completely dry the grains or reach a steady state on the MR.

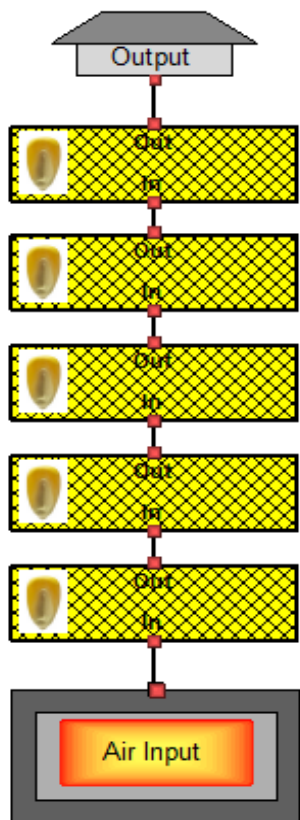


Figure 2. Example of stacked thin corn layers blocks attached to the *Dryer_Input* and *Air_Exit* blocks

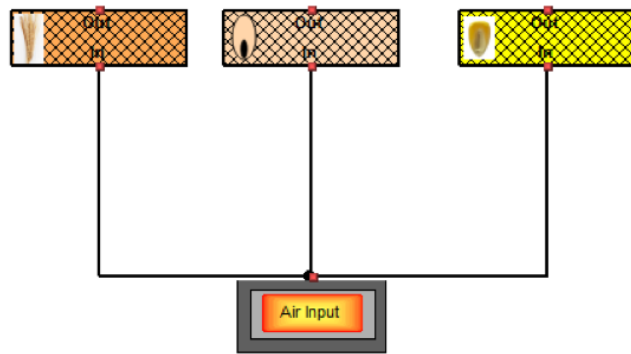
3.2 Deep bed simulation for corn

The MR was similar at the beginning of the drying; however, the difference between the layers increased as the drying continued, as observed in Figure 4. At the end of the simulation, the difference between the moisture content of the grains in the first and last layers was of 2 percentage points. This demonstrates that, when the air crossed the last layer, it was already carrying moisture from the other layers, which decreased the capacity of the air to remove water from the grains at the end of the dryer. Comparing the humidity ratio (HR) during the time for these two layers, reiterates how the air was filled with moisture and had its drying capacity reduced at the end of the grain column (Figure 5). While air travels through the column of grain, it absorbs water vapor from the grains and increases its HR. As expected, the humidity ratio was higher at the beginning of the drying simulation and decreased over time since the grain had less water to be absorbed. The changes for HR over time for the first layer are less perceptible compared to the last one, indicating that the humidity ratio exiting the last layer is clearly higher than the first layer's HR.

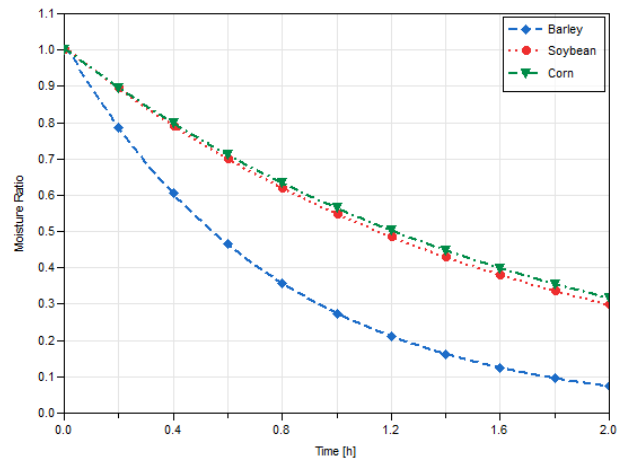
An *AirSensor* component was connected after the last layer of the deep-bed simulation to measure the psychrometric properties of the air. Figure 6 shows three air properties at the exit of the dryer: saturated (p_{ws}) and partial (p_w) pressure of water vapor, and Relative Humidity (RH). It can be observed that the p_{ws} was nearly constant (4.23 kPa) during the simulation, which means the air was fully saturated with water vapor at the exit of the last layer for the entire drying time. Meanwhile, the p_w had an increase before descending until the end where it reached a final value of around 1.90 kPa. That is, the water vapor molecules were exerting less pressure in moist air. This can be associated with the removal of moisture from the grains with time and less free water for the dry air to absorb. RH is the ratio of water vapor in the air and the pressure when it is saturated, i.e., it can be expressed in terms of p_w/p_{ws} . Similarly to the p_w trend, the RH had a sudden raise before decreasing, as observed in Figure 6. The final value for RH was 0.449, corresponding to the ratio p_w/p_{ws} at the end of the simulation.

3.3 Temperature Comparison

The air temperature has a significant role during the drying process of biological materials. It will affect the rate of the process and the equilibrium moisture content of the product, as observed from Equations 1 to 4, and 6. The simulation results for MC are illustrated in Figure 7. As expected, the higher the air temperature, the faster the grain dried. The layer of corn reached MC of less than 5% in three hours, under air temperature of 90 °C. Meanwhile, the layer with the same characteristics, but with the air temperature at 25 °C, achieved a final MC value around 15%. Even though the models results were consistent with what was predicted, the simulation results needs to be val-



(a)



(b)

Figure 3. Thin layers connected to the dryer (a) and Simulation results (b) for Barley, Soybean, and corn

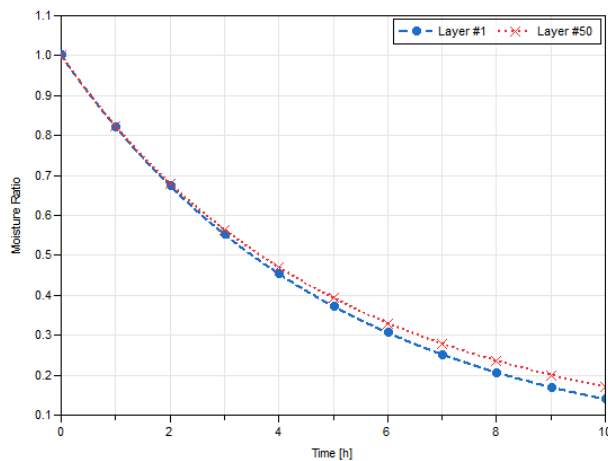


Figure 4. Moisture Ratio over time for the first and last layer of a deep bed simulation

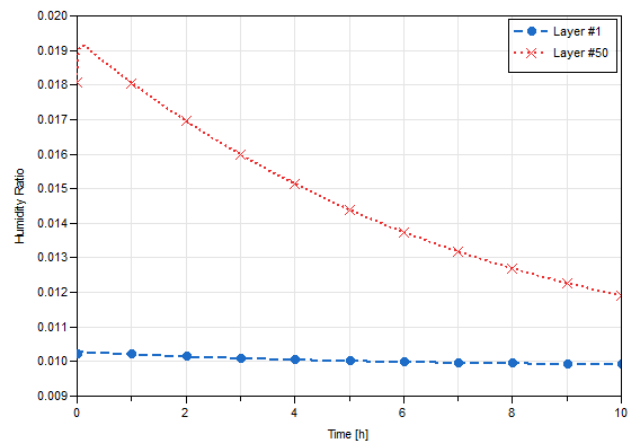


Figure 5. Humidity Ratio (m/m) at the exit of the first and last layer over time for the a deep bed simulation

idated with experimental results to verify the precision of the Modelica library.

There is an extensive discussion about the use of high temperatures for faster drying over the risk of losing grain quality due to heat damage or loss of dry mass. This could be used as a comparative tool to real situations to make the best decisions about the amount of time for the grain to dry.

4 Model Validation

The simulated results were compared to experimental data from different sources for each of the grains studied (Markowski et al., 2010; Li and Morey, 1984; Freire et al., 2005). The products features and the characteristics of the drying process are shown in Table 2. The reason why these studies were chosen was because they had similar characteristics to the designed Modelica library, such as a broad temperature range, thin-layer of grains, and complete de-

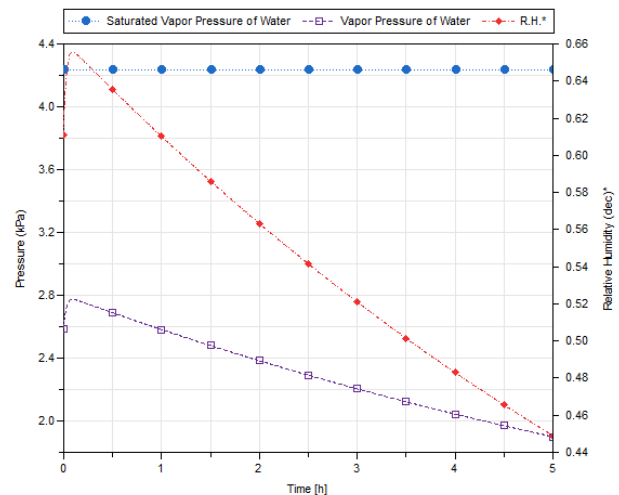


Figure 6. Saturated and partial Water Vapor pressure (left axis), and Relative Humidity (right axis) of the air at the dryer exit

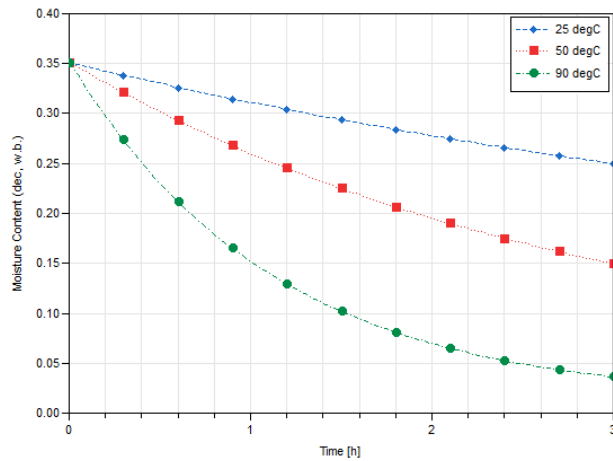


Figure 7. Moisture content for thin corn layers at different drying temperatures

scription of the utilized drying process.

The *Modelica* results compared to the experimental data from different studies can be seen in Figure 8. The experimental data are similar to the simulated results. In general, they followed the same drying trend, but they did not have the exact final value. While the results for barley and soybean are in Moisture Ratio (Figure 8 (a) and (c)), the results for corn were reported in terms of the product Moisture Content (Figure 8(b)). To avoid misleading and erroneous interpretation, the simulation results were reported using the variable as the extracted experimental data.

There are some differences that can be attributed to variations in grain composition, drying management, and final moisture determination. For example, different varieties of the same grain have distinct shapes that influence the drying rate. Similarly, the drying geometry and method vary and may affect the final moisture content. Additionally, error accumulates during time of both data; thus, the final MR or MC value is impacted by the error of the entire process. Table 3 shows the mean squared error (MSE), the average relative error (RE), and the relative error for the final moisture content or moisture ratio of the drying between the experimental and simulated data. It can be observed from this table that the lowest relative error for corn, barley, and soybean were respectively at 48, 27, and 31.5 °C. Nearly the same is applied to the final RE, the exception is corn at 93 °C. For the MSE, the min-

imum values were at 48, 93, and 58.5 °C for barley, corn, and soybean, respectively. Between all grains, barley had the lowest values; thus, the *Modelica* model could estimate moisture better for this grain. The final RE is biased since the error is accumulated from the beginning of the simulation. In general, this value was higher than the average RE (9 out of 12 cases), which showed how the error was carried along during the simulation.

It is difficult to validate the model with so many variables to control. These differences could reflect in the difference between experimental and simulated data. Additionally, there are parameters that depend on grain chemical composition that can change depending on the variety of the product. For example, some corn kernels with less protein and more starch content could have different drying curves. Also, more experimental data could be obtained to recalibrate some empirical parameters for the grains used. Also, experimental data could optimize the model to achieve better results from the simulations. However, every new grain variety, with different composition and geometry, could lead to distinct sets of constants to be used in mathematical modeling of a drying process.

5 Conclusion and Future Work

Several *Modelica* components were modeled to simulate the grain drying process. Barley, corn, and soybean were studied to be used on this model, where the simulation results were compared to experimental data from different sources. There were some differences between these two types of data that can be attributed to differences in grain composition, controlling the drying environment, and errors associated with using some outdated empirical parameters. Even though *Dymola* was used for modeling and simulation, the library can be used with other open-source environments for *Modelica*. Overall, this *Modelica* library could be an educational accessory to learners interested in this topic.

This library can be further expanded to simulate more grains and other agricultural products. Likewise, for this study, only a fixed-bed dryer was simulated; thus, a variety of dryers could be added to this set of components to broad its use. Some additional capabilities could be added such as variable weather circumstances and mold controlling through grain and air conditions. The *Modelica* capabilities to be applied to agriculture are abundant, and it has great potential to be further explored.

Table 3. Mean squared error and average relative difference between experimental and simulation data for the three grains

| | Barley | | | | Corn | | | | | Soybean | | |
|--|--------|--------|---------|--------|------|------|------|-------|-------|---------|--------|--------|
| | 33 | 41 | 48 | 56 | 27 | 49 | 71 | 93 | 116 | 31.5 | 45 | 58.5 |
| MSE (dec. ² or % ²) | 0.0026 | 0.0033 | 5.21e-4 | 0.0020 | 1.09 | 4.40 | 1.37 | 0.186 | 0.982 | 0.0050 | 0.0037 | 0.0027 |
| Average R.E. (%) | 8.57 | 12.1 | 5.91 | 15.6 | 5.86 | 22.7 | 21.1 | 7.03 | 17.9 | 9.38 | 17.6 | 21.3 |
| Final R.E. (%) | 12.4 | 24.9 | 8.53 | 24.2 | 8.32 | 54.5 | 56.8 | 0.33 | 8.21 | 6.15 | 37.5 | 48.2 |

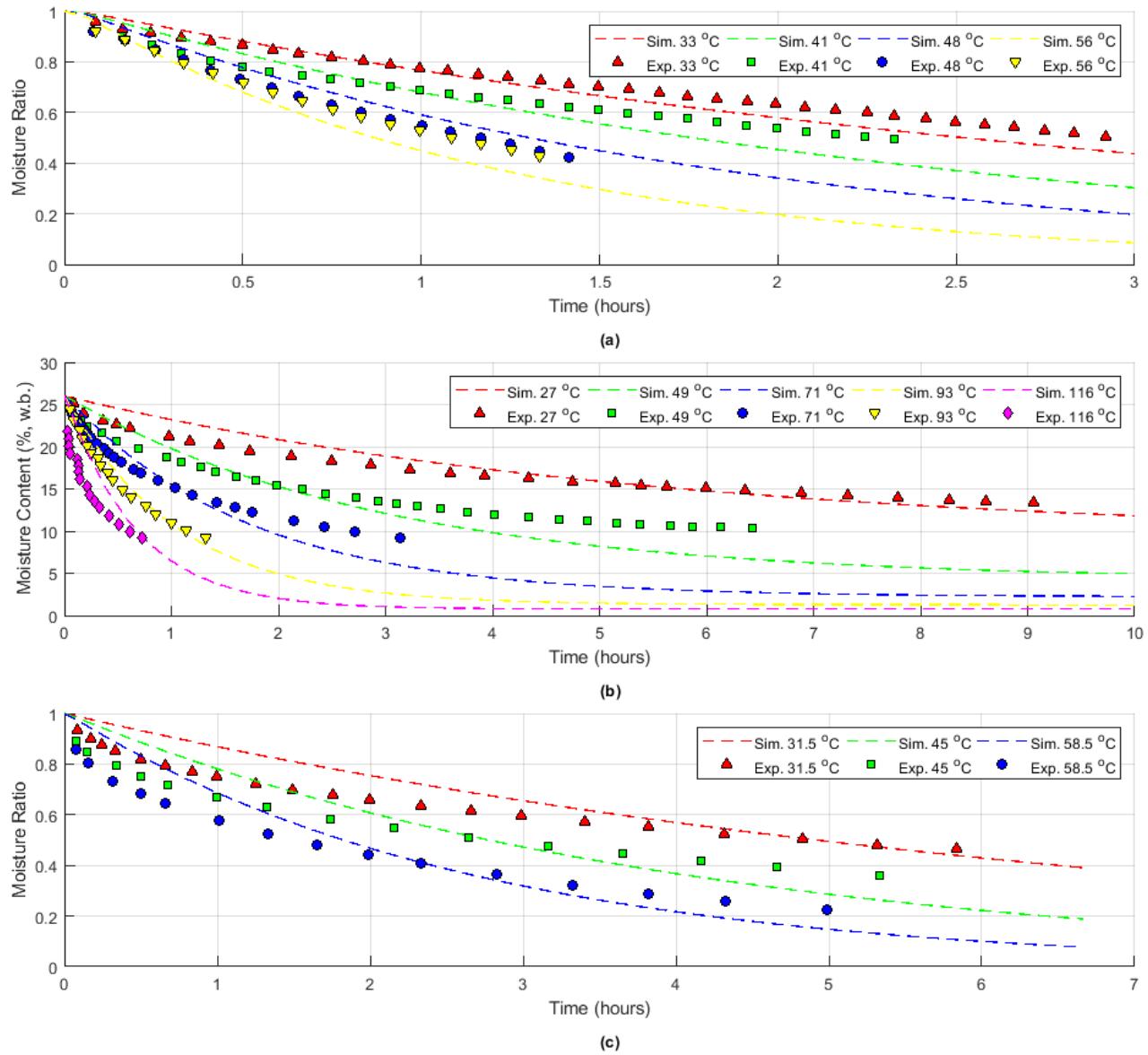


Figure 8. Results comparison between experimental and simulation data for Barley (a), Corn (b), and Soybean (c)

References

- Antoine Aurousseau, Valery Vuillerme, and Jean-Jacques Beziau. Modeling of Linear Concentrating Solar Power using Direct Steam Generation with Parabolic-Trough. pages 595–603, 9 2015. doi:10.3384/ecp15118595. URL <http://www.ep.liu.se/ecp/article.asp?issue=118%26article=64>.
- Jannatul Azmir, Qinfu Hou, and Aibing Yu. Discrete particle simulation of food grain drying in a fluidised bed. *Powder Technology*, 323:238–249, 2018. ISSN 1873328X. doi:10.1016/j.powtec.2017.10.019. URL <https://doi.org/10.1016/j.powtec.2017.10.019>.
- G. R. Baughman, M. Y. Hamdy, and H. J. Barre. Analog Computer Simulation of Deep-Bed Drying of Grain. *Transactions of the ASAE*, 14(6):1058–1060, 1971. ISSN 2151-0059. doi:10.13031/2013.38452. URL <http://elibrary.asabe.org/abstract.asp?JID=3&AID=38452&CID=t1971&v=14&i=6&T=1>.
- Luis A. Bortolaia, Oleg Khatchatourian, and Horacio A. Vielmo. Analysis of soybean drying dynamics in thin layer. *13th Brazilian Congress of Thermal Sciences and Engineering December 05-10, 2010, Uberlandia, MG, Brazil*, 2010.
- Donald B. Brooker, Fred W. Bakker-Arkema, and Carl W. Hall. *Drying and storage of grains and oilseeds*. Van Nostrand Reinhold, 1992. ISBN 9780442205157. URL <http://www.springer.com/us/book/9780442205157>.
- D.M. Bruce. Exposed-layer barley drying: Three models fitted to new data up to 150°C. *Journal of Agricultural Engineering Research*, 32(4):337–348, 12 1985. ISSN 0021-8634. doi:10.1016/0021-8634(85)90098-8. URL <https://www.sciencedirect.com/science/article/pii/0021863485900988>.
- Clyde M. Christensen and H. H. Kaufmann. Deterioration of Stored Grains by Fungi. *Annual Review of Phytopathology*, 3(1):69–84, 9 1965. ISSN 0066-4286. doi:10.1146/annurev.py.03.090165.000441. URL <http://www.annualreviews.org/doi/10.1146/annurev.py.03.090165.000441>.
- Fabio B. Freire, Marcos A.S. Barrozo, Dermeval J.M. Sartori, and Jose T. Freire. Study of the drying kinetics in thin layer: Fixed and moving bed. *Drying Technology*, 23(7):1451–1464, 2005. ISSN 07373937. doi:10.1081/DRT-200063508.
- Rahul Jain, Kannan M. Moudgalya, Peter Fritzsøn, and Adrian Pop. Development of a Thermodynamic Engine in Open-Modelica. pages 89–99, 7 2017. doi:10.3384/ecp1713289. URL <http://www.ep.liu.se/ecp/article.asp?issue=132%26article=009>.
- C.-C. Jia, W. Yang, T. J. Siebenmorgen, and A. G. Cnossen. Development of Computer Simulation Software for Single Grain Kernel Drying, Tempering, and Stress Analysis. *Transactions of the ASAE*, 45(5):1485–1492, 2002. ISSN 2151-0059. doi:10.13031/2013.11039. URL <http://elibrary.asabe.org/abstract.asp?JID=3&AID=11039&CID=t2002&v=45&i=5&T=1>.
- O. A. Khatchatourian, H. A. Vielmo, and L. A. Bortolaia. Modelling and simulation of cross flow grain dryers. *Biosystems Engineering*, 116(4):335–345, 2013. ISSN 15375110. doi:10.1016/j.biosystemseng.2013.09.001. URL <http://dx.doi.org/10.1016/j.biosystemseng.2013.09.001>.
- J. Lacey. Pre- and post-harvest ecology of fungi causing spoilage of foods and other stored products. *Journal of Applied Bacteriology*, 67:11s–25s, 12 1989. ISSN 00218847. doi:10.1111/j.1365-2672.1989.tb03766.x. URL <http://doi.wiley.com/10.1111/j.1365-2672.1989.tb03766.x>.
- Huizhen Li and R Vance Morey. Thin-layer drying of yellow dent corn. *Transactions of the ASABE*, 27(2):581–585, 1984. ISSN 00012351. doi:10.13031/2013.32832. URL <http://elibrary.asabe.org/abstract.asp?aid=32832&t=3>.
- Marek Markowski, Ireneusz Białobrzewski, and Agnieszka Modrzewska. Kinetics of spouted-bed drying of barley: Diffusivities for sphere and ellipsoid. *Journal of Food Engineering*, 96(3):380–387, 2 2010. ISSN 0260-8774. doi:10.1016/J.JFOODENG.2009.08.011. URL <https://www.sciencedirect.com/science/article/pii/S0260877409004099?via%3Dihub>.
- Lamber Otten and George Samaan. Determination of the Specific Heat of Agricultural Materials: Part II. Experimental Results. *Canadian Agricultural Engineering*, 22(1):25–27, 1980.
- S. Pabis, D. S. Jayas, and S. Cenkowski. *Grain Drying: Theory and Practice*. John Wiley and Sons Ltd, New York, United States, 1989. ISBN 0471573876.
- USDA. *Grain Inspection Handbook - Book II*. Washington, DC, 2013. URL <https://www.gipsa.usda.gov/fgis/handbook/BK2/BookII4-11-2017.pdf>.
- J.S. Wall, C. James, and G.L. Donaldson. Corn proteins: chemical and physical changes during drying of grain. *Cereal chemistry*, v. 52(no. 6):779–790, 1975.

Modelica library for the systems engineering of railway brakes

Marc Ehret

Institute of System Dynamics and Control, German Aerospace Center, Germany, marc.ehret@dlr.de

Abstract

This work outlines the role of system simulation for the development process of railway brakes. The principles of systems engineering motivate the use of computer based simulation in order to enhance the understanding of systems and to verify the behavior of systems in early design phases. For this reason, the Modelica library "Virtual Train Brakes" is presented which enables engineers to generate simulation models of railway brakes and to perform system simulations during different phases of the development process. By modeling and simulating the brake systems of a passenger and a freight train, the capability of the library is demonstrated and further investigations are motivated.

Keywords: Railway Brakes, Systems Engineering, Multi-Level Models, Variant Models, Generic Models

1 Introduction

Railway brakes are complex technical systems that fulfill high requirements concerning safety, availability and reliability. These systems are constructed to be deployed in a difficult operational environment, characterized by large temperature ranges and heavy dynamic loads during long utilization times. Furthermore, a huge amount of kinetic energy needs to be converted during a single brake application which requires that brake discs are able to absorb up to 30 MJ of energy (Breuer, 2006). Beside severe operational conditions, the diversity of railway vehicles and brake types demands to design individual adapted brake systems which leads to a high variety and individuality of the developed systems. This individuality and the corresponding variety of brake systems is in contrast to the comparatively low number of items that is usually delivered (Anton, 2010). In order to reduce technical as well as economical risks the application of systems engineering is indispensable for of the development process of railway brakes.

Systems engineering is an approach that enforces system architects to deduce physical solutions by identifying stakeholders, specifying their requirements and map them by system functions. Thereby, an interdisciplinary and holistic view of the desired system is generated, which is deepened throughout the design process. By this means, mistakes are identified and eliminated in early design phases before costly changes during the implementation are necessary. Due to the increasing complexity and functionality of technical systems arising from the increasing

content of electronic and software components, systems engineering has become a wide spread and important procedure in product development (INCOSE, 2015).

In the context of systems engineering, computer based system simulation is a powerful tool since it supports designers to understand the behavior of systems in design stages where an experimental analysis is not feasible. Furthermore, it allows engineers to check their own thinking, to analyze alternatives and capabilities of the system and to communicate their concept to others (INCOSE, 2015) (Mittal et al., 2017). Modelica is a well suitable modeling language to generate models of multi-physical technical systems, such as railway brake systems and to study their physical behavior.

The goal of this work is to develop the concept of a Modelica library which provides an environment for the application of system simulation throughout the entire development process of railway brakes. As shown in Figure 2, the main tasks of the library are:

- support dimensioning
- analyze and optimize system behavior
- support integration
- support system test and commissioning

Initially, this work introduces railway brake systems and the role of systems engineering in the scope of their design. Subsequently, use cases of system simulation during the development process are discussed which are the basis for the structure and implementation of the library with Modelica in Dymola. Varying requirements concerning the accuracy and the computational effort of the models are considered and the generic composition and diverse configuration variants of railway brake systems are respected. After the presentation of the library it is applied to model and simulate the brake systems of a passenger and a freight train. The development of the library is an ongoing work. This paper is primarily meant to motivate its usage and to present the modeling concept. An outlook is given which states the current limits of the library and motivates further investigations.

2 Systems engineering of railway brakes

2.1 Introduction to railway brakes

The main functions of railway brake systems are (i) the conversion of kinetic energy in order to reduce the velocity of the train, (ii) to counteract the downhill-slope force in order to keep the velocity of the train and (iii) to prevent a stationary train of moving due to gradients or other external forces, for instance caused by wind (Knorr-Bremse, 2003).

The basic safety requirements demand that all types of railway vehicles have an automatic, continuous and inexhaustible brake system. This means that the brake systems of all cars of a railway vehicle are controlled by a through signal line (continuity) and that in case of an error in this signal line caused by leakage or cutoff, each of the cars of the vehicle stops automatically. Inexhaustibility requires that the performance of the brake system is still available, although there have been repeated brake applications before (DIN EN 14198, 2005) (DIN EN 15734-1, 2013). The brake distance in case of emergency braking is the major performance requirement brake systems have to meet. The maximum brake distances of locomotives and passenger trains required by (Commission Regulation (EU) No 1302/2014, 2014) is shown in Table 1 relating to different velocities. Additionally, the mean value of the corresponding deceleration is given assuming a constant deceleration. This value does not include the delay and response time of the brakes as well as the velocity-dependent adhesion between wheel and rail limiting the maximum feasible deceleration described in chapter 4.2.4.5.2 and 4.2.4.6.1 in (Commission Regulation (EU) No 1302/2014, 2014).

Table 1. Brake distance requirements for emergency braking of locomotives and passenger trains relating to different initial velocities according to (Commission Regulation (EU) No 1302/2014, 2014)

| Initial velocity $\left[\frac{km}{h}\right]$ | Required brake distance [m] | Mean deceleration $\left[\frac{m}{s^2}\right]$ |
|--|-----------------------------|--|
| 350 | 5360 | 0.88 |
| 300 | 3650 | 0.95 |
| 250 | 2430 | 0.99 |
| 200 | 1500 | 1.03 |

In order to fulfill these requirements and functions railway vehicles are equipped with different types of brake systems, shown in Figure 1, depending on the type of vehicle (passenger cars, freight cars, locomotives) and its operating modes (service brake, emergency brake, parking brake, holding brake).

Due to its high level of safety all railway vehicles are at least equipped with frictional brake systems, such as

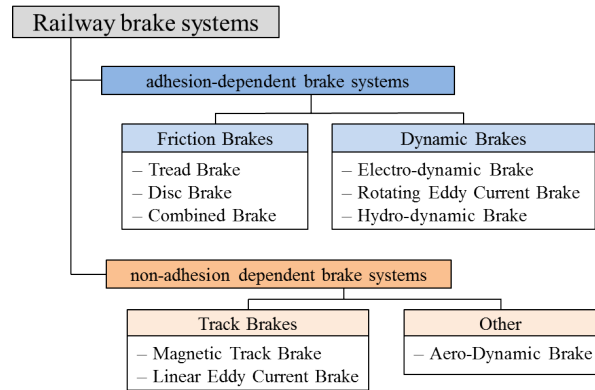


Figure 1. Classification of railway brakes according to (Knorr-Bremse, 2003)

tread and disc brakes, which are mostly activated by compressed air except for trams which use hydraulic media due to the limited available space in these vehicles. Additional brake systems, e.g., electro-dynamic brakes in electrically driven vehicles and hydro-dynamic brakes in diesel-hydraulic vehicles, are applied to serve as service brakes and to support the frictional brakes in order to minimize wear and thus extend the technical lifetime of brake systems. Non-adhesion dependent systems, for instance track-brakes, are not limited by the maximum transferable force between rail and wheel. They are deployed to minimize the braking distance in case of an emergency. The faultless cooperation and redundancy of the diverse subsystems is an important aspect for the safe, secure and reliable operation of trains.

2.2 Systems engineering

Depending on the type of system there are different process models which determine the design procedure of systems. Haberfellner and Daenzer differentiate between plan-driven methods, such as the "V-Model" or the "Waterfall-Model" and agile methods, such as "Scrum" or the "Spiral-Model" (Haberfellner and Daenzer, 2002). The latter methods are commonly used for software engineering in which flexible and less sequential frameworks are preferred due to the dynamic and changing environment throughout the development process. In contrast, plan-driven methods are characterized by fixed steps and defined sequences during the development process. This is essential for the design of large multi-physical systems which are subject to high requirements regarding safety and reliability, such as railway brake systems.

The most common plan driven method is the "V-Model", as shown in Figure 2. The basic idea is to follow a structured top-down development process. At first the stakeholders are identified, who are the source of the system requirements from which in turn the system specifications are deduced (Haberfellner and Daenzer, 2002) (INCOSE, 2015). During the top-down development process the specifications of the system are disassembled into

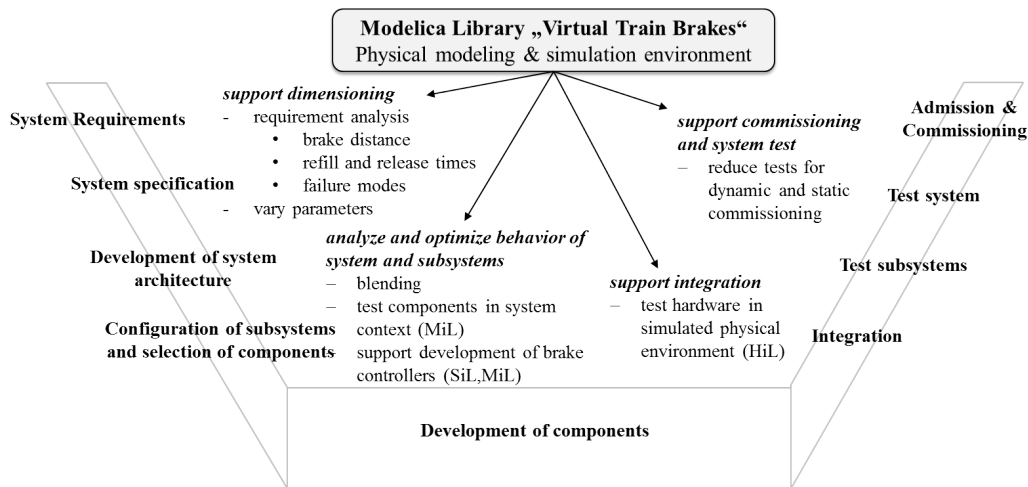


Figure 2. "V-Model" as development process of railway brake systems with use cases for system simulation

sub-specifications of subsystems until the smallest entity is reached. In each stage the physical architecture, system elements and interfaces are derived from the specifications using context and functional analysis. During the bottom-up implementation and integration phase the elements and subsystems are built and verified, if they fulfill the defined specifications. Finally, this leads to the integration, verification and validation of the entire system in its operational environment.

The development of railway brakes is a typical case of application engineering, since brake systems consist of standard components provided by an existent product asset, such as brake cylinders, typical valves and pre-assembled subsystems like integrated modules for brake control (André et al., 2017). Application engineering is a particular type of systems engineering in which the elementary components and subsystems are predefined parts of a component portfolio. Many technical systems are designed by reusing predefined and for the system characteristic components which are selected, configured and interconnected.

2.3 Simulation of railway brakes

To analyze the time dependent behavior of railway brakes physical models can be applied, which are represented by mathematical equations deduced from natural laws. The object-oriented and component-based modeling approach of Modelica is well suited to develop these physical models and is therefore used in this work. The goal of the Modelica library "Virtual Train Brakes" is to provide physical models for system simulation throughout the entire systems engineering process of railway brakes shown in Figure 2.

As stated in (Anton, 2010), the ambivalence of cost for design and quality of the designed system in scope of the tender process of railway brakes strongly motivates the use of system simulation. During the tender process system engineers design the rough architecture of brake systems. In this phase the type of applied brake systems

as well as the number of brakes and their dimensions are defined depending on the specification of the customer. Modeling and simulating the designed architectures allows to gather useful information regarding the system behavior, such as the effects of refill and release times or failure modes on the braking distance. Furthermore, the sensitivity and impact of varying system parameters, e.g., the size of reservoirs and diameters of pipes, can be analyzed. Thus, system simulations help to avoid oversizing of systems and to submit competitive offers during the tender process.

As mentioned in 2.1 railway vehicles are usually equipped with several types of brake systems. To achieve the desired deceleration while minimizing wear a complex brake management is necessary. The so called blending defines the contribution of the different brake systems relating to the current operating mode. System simulation facilitates the analysis and review of the individually designed blending concepts long before commissioning. Furthermore, it allows designers to apply numerical optimization in order to find the ideal blending concepts.

Another application of system models is to simulate the physical environment of components or subsystems. This enables system designers to study and review the behavior of components or subsystems in the context of their environment. During the development of electronic devices, such as control units, a model representing the basic control algorithm ("Model in the Loop": MiL), the implemented software ("Software in the Loop": SiL) and finally the hardware of the unit ("Hardware in the Loop": HiL) are designed and verified by integrating them into a simulated physical environment as described in (Tischer and Widmann, 2012). Depending on the integrated object different requirements concerning the computation rate, model interfaces and the accuracy of the simulation need to be considered. The application of HiL-testing for the development of railway brake systems is demonstrated in (Pugi et al., 2006), (Kang et al., 2009) and (Lee and Kang, 2015).

Finally, virtual testing with validated system models allows system engineers to reduce the efforts of experimental testing which is directly linked to time and costs for implementation, integration and commissioning. The main intention of virtual testing is not to replace experimental tests, but to narrow the quantity of tests which are necessary to adjust the detailed setting of components, e.g., valves and nozzles, in order to harmonize and verify the entire system behavior.

To provide physical models throughout the entire development process a model library is required which bases on the component and system portfolio of railway brakes. Furthermore, the presented scenarios for system simulation demand models which are customizable with respect to the level of detail, accuracy and computational effort depending on the scope of simulation.

3 Modelica library "Virtual Train Brakes"

The Modelica library "Virtual Train Brakes" is an approach to realize the desired simulation environment. Initially, the structure and modeling concepts are introduced followed by the implementation in Modelica.

3.1 Library structure and modeling concepts

The structure of the library is pictured in Figure 3. The foundation is the component library which contains multi-physical models of typical technical components analog to the product asset of railway brakes, for instance distributors valves, pressure transformers or brake cylinders. These technical components consist of core elements from different physical domains, such as nozzles and volumes in case of pneumatic components. In this work core elements from the Modelica Standard Library and from the Pneumatics Library developed by Modelon, described in (Modelon AB, 2010) and (Beater, 2007), are applied. To fulfill the requirements of adaptable accuracy and computation rate the models of the technical components are implemented as multi-level models. This means that different levels of detail of a component are depicted by selectable level-models which share the same physical interfaces but differ in their modeling content.

The technical components provide the elements for the system library. This library contains generic models of systems and subsystems of railway brakes what eases the generation of large models. The pre-assembled system models result from the definition and separation of brake systems into subsystems, including boundaries and interfaces analog to the top-down development process of the "V-Model". In case of the system "train", which is exemplarily subdivided in Figure 4, the system model in the highest layer (Layer 1) consists of the subsystems "car". This contains the subsystems "brake panel", "bogie pneumatic" and "bogie mechanic" which in turn include interconnected technical components, e.g., distributor valves and the pressure transformers. In this generic concept

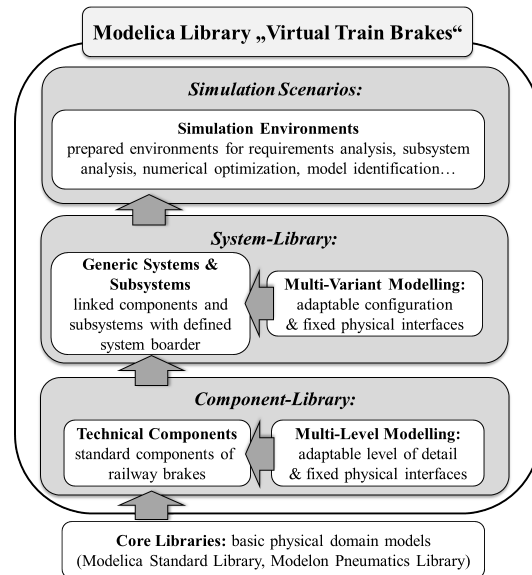


Figure 3. Structure of the library "Virtual Train Brakes"

components can be part of any subsystem in any layer.

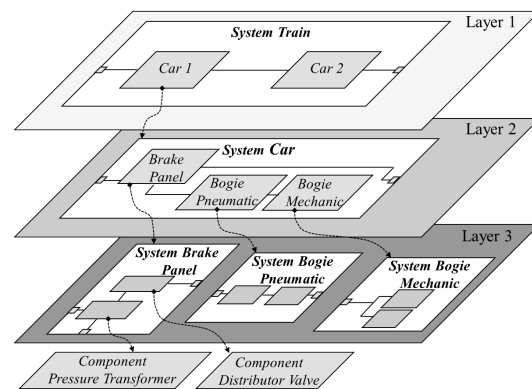


Figure 4. Exemplary scheme of the generic system "train"

Railway brake systems are individually designed depending on the varying specifications and on the vehicle they are integrated into. This leads to a broad diversity of designed systems and it is unfeasible to provide generic models for all them. Nevertheless, particular subsystems are combined to modular platforms for specific product families of which certain variants exist (André et al., 2017). By identifying and modeling these standard variants it is possible to ease the model generation for a large amount of railway brake systems. For this reason, the generic models of the system library are implemented as multi-variant models differing in their configuration but sharing the same physical interfaces, as exemplarily shown in Figure 5 for the subsystem "brake panel". The restriction of having the same interfaces is indispensable. Otherwise the integration of a configurable subsystem into a higher layer could fail due to violation of the interconnections between a configured subsystem and the elements in its environment. Furthermore, the complexity of the generic models is limited and an extensive configu-

ration management is avoided.

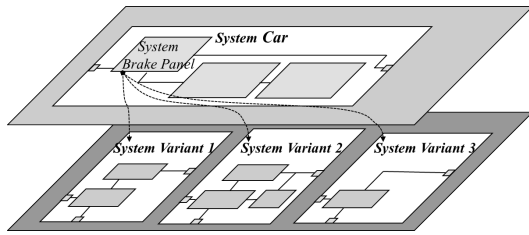


Figure 5. Exemplary scheme of variant modeling in a generic system structure

3.2 Implementation of the library in Modelica

According to the presented structure the Modelica library is subdivided into a component library and a systems library in which the models are stored. In the following, the implementation of component models and system models is described.

3.2.1 Modeling multi-level components

The implementation of a technical component is described by using the example of the component "Brake Cylinder". The corresponding component package contains a so-called container model in which two instances of the models "BrakeCylinderLevel_1" and "BrakeCylinderLevel_2" exist in parallel, as shown in Figure 6 and suggested by (Kuhn et al., 2008). They represent two different modeling approaches which basically differ in their level of detail. Both level-models are connected to the ports of the container model where they are implemented by using conditional statements. This means that the models are not considered when translating the container model, if the corresponding conditional statement is false. The activation of the desired model is controlled by an Integer parameter called "Level" which activates "brakeCylinderLevel_1" in case of a value equal to 1 and "brakeCylinderLevel_2" for a value equal to 2. Thus, the selection of the desired model is controlled by a single parameter and does not have to be carried out manually as it would be done in case of replaceable models (Kuhn et al., 2008).

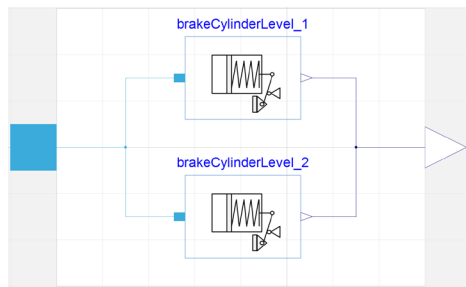


Figure 6. Container model of the component "Brake Cylinder" with two conditionally implemented level models

The container model includes an instance of the record "ParameterBrakeCylinder". This record contains the parameter "Level" and instances of individual records of

the level-models "ParameterBrakeCylinder_Level_1" and "ParameterBrakeCylinder_Level_2" leading to a clear separation of the model-dependent parameter sets.

To ensure that the container model and the level-models are compatible with each other they are inherent classes of the class "BrakeCylinderTemplate". This model defines the interfaces and the icon of the component model. Thereby, the component models represent the same ports, regardless of the selected level-model. This is important for a clean integration of component models into system models.

Additionally, the component package provides a simulation environment "SimulateBrakeCylinder" which integrates the model into a testing environment. This is helpful for testing newly developed models and serves as simulation example when applying the component, respectively. Furthermore, it is possible to compare simulation results with experimentally measured data for the verification of the behavior of the model, as shown in 4.1.5. In this case it is important to ensure that the simulation environment represents the initial and boundary conditions which existed during the corresponding experiment.

Figure 7 and 8 show the content of the two level-models of a brake cylinder which convert pneumatic pressure into a clamping force. In model "brakeCylinderLevel_1" the pneumatic input port on the left hand side is connected to a constant pneumatic volume. The pressure in this volume is used to calculate the resulting clamping force depending on parameters, e.g., piston diameters and efficiency by using a simple formula. In contrast, "brakeCylinderLevel_2" uses mechanical and pneumatic components, such as a pneumatic cylinder, a counteracting spring, the brake rigging and a gap element to model the contact between disc and brake pad. This represents a more detailed model of the component taking further physical effects into account, for instance friction, contact effects and pressure dependent volumes.

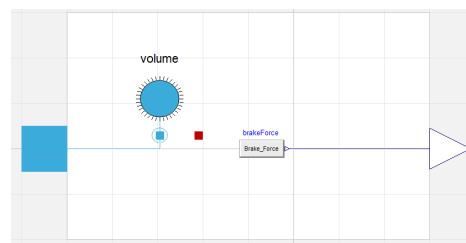


Figure 7. Model content of "BrakeCylinderLevel 1" with volume model from the Modelon Pneumatic Library

Figure 9 shows the impact of the two different modeling approaches on simulation results by comparing the calculated clamping force in case of a pressure gain in the cylinder. The simulation results of the model "brakeCylinderLevel_2" contain transient effects at the beginning. They are caused by friction in the pneumatic cylinder and the contact of pad and disc. These effects might be of interest in case of a detailed investigation, for example the

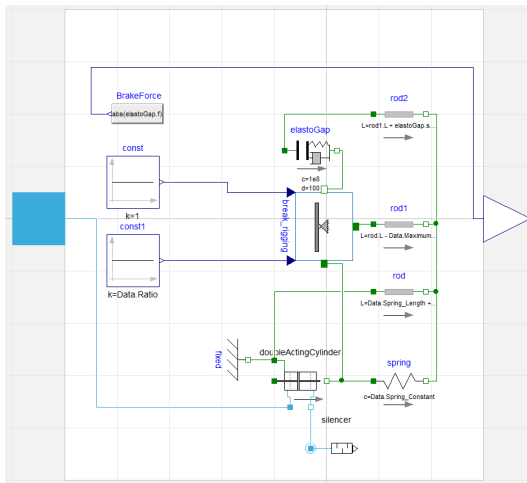


Figure 8. Model content of "BrakeCylinderLevel 2" with cylinder model of the Modelon Pneumatic Library

behavioral analysis of the brake cylinder during the application of the wheel slide protection. Model "BrakeCylinderLevel_1" neglects these effects, but provides the same stationary clamping force. This is sufficient for the analysis of the entire brake system, for instance in case of the estimation of the brake distance.

A comparison of the computation time in Figure 10 shows that the simulation of model "BrakeCylinderLevel_1" (2 states, 0.03 seconds) is 5 times faster than the simulation of model "BrakeCylinderLevel_2" (9 states, 0.16 seconds). Especially the computation of the transient oscillations after 0.2 demands a large computational effort. The bigger the system of interest, the stronger the entire computation rate is influenced by the computational effort for single components. This example emphasizes the need for different model approaches due to the varying scope of simulations described Figure 2.

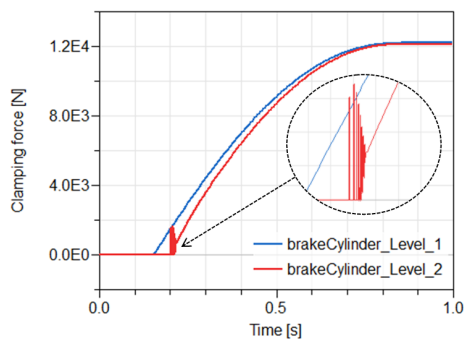


Figure 9. Comparison of simulation results of the level-models "brakeCylinder_Level_1" and "brakeCylinder_Level_2" with zoomed transient effects

3.2.2 Modeling multi-variant and generic systems

To model multi-variant system models, container models are applied analog to the modeling approach of multi-level components described in 3.2.1. The use of container mod-

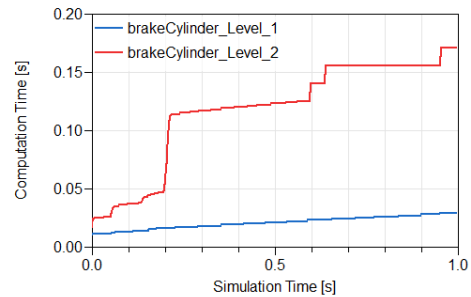


Figure 10. Comparison of computation time of the models "brakeCylinder_Level_1" and "brakeCylinder_Level_2"

els in place of replaceable models enables the user to configure system models by supplying parameters instead of choosing the desired variant manually. The essential difference between the models of systems and components is that the container model a system includes variant-models. They represent different configurations of a system instead of different modeling approaches concerning the level of detail. Thus, the structure of a system package is analog to the component package. Since the parameter sets of different system configurations are mostly the same, there are no individual records for each of the variant-models in contrast to the level-models of components.

To cover further configuration types of a system, integrated components and subsystems are implemented by using conditional statements. These submodels can be deactivated, if they do not appear in a particular system configuration. In this case it is important to ensure that singularities due to missing states are avoided.

By offering variant-models and the possibility to add or remove particular submodels in these variant-models a broad diversity of system configurations can be represented by using a single container model.

Figure 11 shows the GUI of the record "Parameter-BrakePanel" of the system "Brake Panel". To select the desired system variant the integer variable "Variant" is added. Furthermore, boolean parameters can be found, such as "DV_existent" or "EPC_existent" which activate or deactivate integrated submodels (DV: Distributor Valve, EPC: Electro pneumatic Control).

Beside configuration parameters this record contains records of the integrated sub-models, as shown in Figure 11. The parameter "Level_Submodel" serves as default value for the parameter "Level" in the records of the underlying components. Thus, their level of detail may be controlled globally by a single value. If a sub-model is a generic system itself, the selection of the corresponding table is leading to a record with the same structure. Thereby, the parameters of a generic system are hierarchically ordered analog to the generic system structure. Finally, this approach leads to a single record that contains all parameters of all integrated components and subsystems to parameterize and configure the entire generic system model.

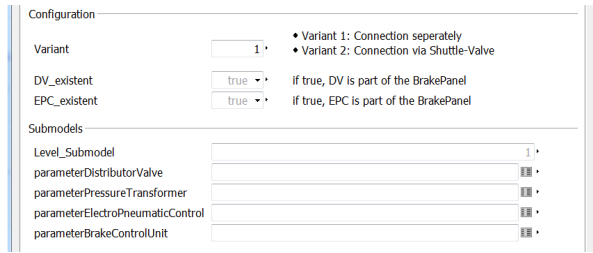


Figure 11. GUI of the record "ParameterBrakePanel"

4 Application scenarios

In the following the library "Virtual Train Brakes" is used to model the railway brake systems of a passenger and a freight train. By performing system simulations according to use cases depicted in Figure 2 exemplary data is generated which serves as basis for the behavioral analysis of the brake systems.

4.1 Modeling and simulating a passenger train

4.1.1 Model generation

Figure 12 presents a generic model of the frictional pneumatic and electro-pneumatic brake system of a passenger train with three railcars. The train is assembled of the submodels "railcar" and "train control" which are pre-assembled generic models of the system library. The cars are connected with pneumatic lines supplying energy and pressure signals to activate the friction brakes and an additional bus line containing electronic brakes signal. To determine the brake distance and velocity of the train, position and speed sensors are connected with mechanical flanges of the railcars.

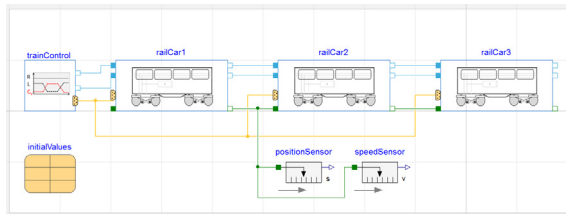


Figure 12. Model of a passenger train with three railcars

Figure 13 depicts the selected variant-model of the first railcar. It contains the subsystems "brake panel", "bogie pneumatic" and "bogie mechanic" as well as components, such as pipes, reservoirs and the pneumatic suspension. The subsystem "bogie mechanic" includes models which calculate the resulting brake forces that decelerate the mass of the train. In these models the friction forces between disc and brake pad are calculated using the clamping forces of the brake cylinders and the corresponding frictional coefficient. These friction forces in turn yield the braking torque acting on the axle depending on the friction radius. The translational braking force is calculated by considering the slip in the contact of wheel and rail resulting from the difference of the rotational ve-

locity of the braked axle and the velocity of the train.

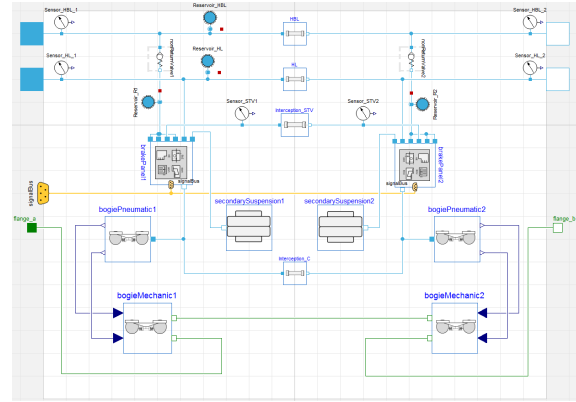


Figure 13. Variant-model of railcar 1

4.1.2 Requirements Analysis

In the train model all bogie masses are connected by flanges leading to a single mass model. This is sufficient to estimate the brake distance for different operating conditions and modes. This kind of analysis is an important step during the dimensioning of the brake system, since the brake distance is the major requirement to be met. The library "Virtual Train Brakes" includes the function "EvaluateBrakingDistanceMatrix" which executes automated system simulations with varying initial values, e.g., the velocity of the train, and parameters, for example the radius of wheels or the axle load.

The function is used to simulate an emergency brake application of the passenger train for several cases differing in the initial velocity and the radius of the wheel. The simulation results of the brake distance over time are depicted in Figure 14. Additionally, the function generates a text document containing the values of the calculated brake distance. This overview enhances the verification of requirements based on the simulation results, as shown in Figure 15. It is shown that the effective wheel radius, which might change over the life time of a train due to wear, has a significant influence on the brake distance. This influence becomes more severe with increasing initial velocity.

4.1.3 Analysis of Refill Time

Beside estimating the brake distance the presented simulations allow to analyze the states and the related behavior of the system. Figure 16 shows the first 3 seconds of the precontrol pressure and the resulting cylinder pressure (C-pressure) for the run $v_0=44.4$ m/s and $R=0.385$ m. The refill time of the C-pressure is directly correlated to the resulting brake distance, since it determines the moment when the braking torque is fully available. It is mainly influenced by the volumes and pneumatic resistances of valves and pipes that provide the brake cylinders with air. In this case the refill time takes about 1.25 seconds, as shown in Figure 16.

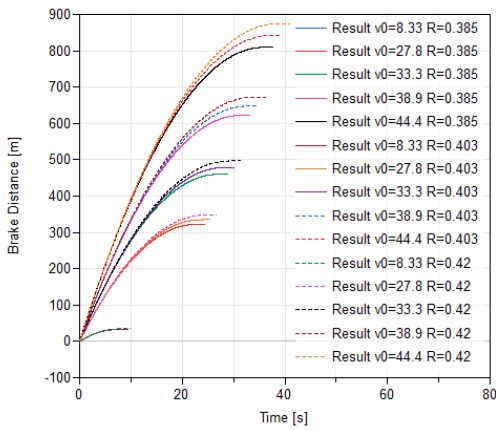


Figure 14. Results of simulations of the generic passenger train model with initial velocity v_0 and radius R of the wheel

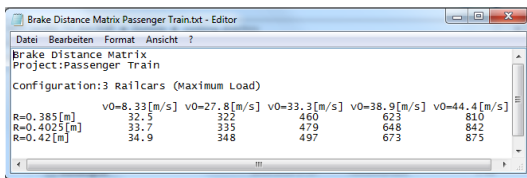


Figure 15. Final brake distances estimated by simulations

4.1.4 Worst-case analysis

The use of numerical optimization enables to perform a worst-case analysis. This type of analysis allows to evaluate the limiting parameters of the brake system, which is very useful during the phases of dimensioning and optimization. An exemplary application is the analysis of the friction coefficient in the contact between disc and brake pad. This parameter strongly influences the brake distance and is dependent on the conditions in the contact and the rotational speed of the disc (Knorr-Bremse, 2003).

By applying the optimization library presented in (Pfeiffer, 2012) in conjunction with the system model of the brake a numerical optimization is performed. The goal is to identify the minimum mean value of the friction coefficient that does not violate the brake distance requirement of 950 meters for an initial velocity of 160 km/h. The start value of the friction coefficient is 0.365 and the maximum range is $[0.1, 0.5]$, respectively. The corresponding flow chart is depicted in Figure 17. By automatically tuning the value of the friction coefficient within the defined interval and performing simulations of the adapted model the optimization seeks for the limiting value which leads to a violation of the required brake distance. Figure 17 shows the table of the log-file containing the final results of the optimization. The optimization detects a minimum mean value for the friction coefficient of 0.335. This value causes a brake distance of 950 meters with a deviation of 0.559 meters which is the value of the minimized criteria $|s - s_R|$ within the interval defined by ϵ . The design of the brake pad and the disc needs to ensure that the components are able to fulfill the detected limit.

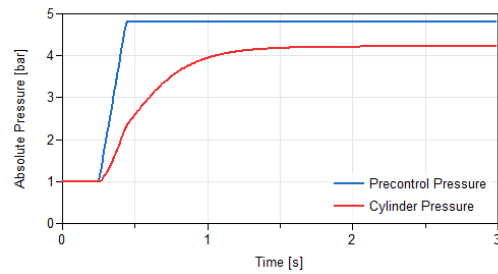
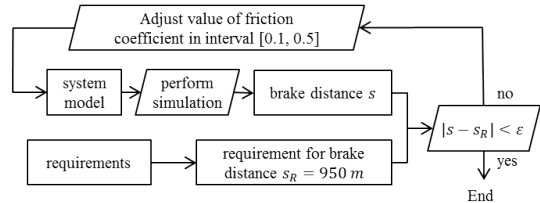


Figure 16. Results of precontrol pressure and resulting cylinder pressure of run $v_0=44.4$ m/s and $R=0.385$ m



| name | value | difference to start | min | max |
|-----------------------|-------|---------------------|-----|-----|
| initialValues.my_fric | 0.335 | -0.03 | 0.1 | 0.5 |

| name | scaled criteria | diff. to start | unscaled criteria | usage | demand value |
|------------------------------|-----------------|----------------|-------------------|----------|--------------|
| requirementCriteria.Criteria | 0.559 | -99.3% | 0.559 | minimize | 1 |

Figure 17. Flow chart of worst-case analysis and final result

4.1.5 Model Identification

Another application of numerical optimization in terms of system simulation is the identification of sensitive model parameters for the validation of models by measurements. The validation of component models is important to generate reliable simulation results of an assembled system.

The component "pressure transformer" influences the fill- and release times of the brake cylinders and thus the brake distance of the train. This relay valve provides the brake cylinders with compressed air from a reservoir and deaerates the cylinders in order to release the brakes (Knorr-Bremse, 2003). The sensitive parameters of the corresponding component model are identified by applying a numerical optimization with data from an experimental component test. The corresponding level-model_1 basically contains two nozzles which determine the dynamic behavior of the component in case of venting and deaerating. The sonic conductance is the sensitive parameter of a pneumatic resistance influencing the mass flow at a particular pressure ratio (Beater, 2007). By varying the sonic conductance of these nozzles in scope of a numerical optimization, the model is tuned to minimize the integrated squared deviation of measurement and simulation results. The default value of the sonic conductance of the nozzles is $10^{-7} [m^3/sPa]$ and the boundaries for the optimization are set to 10^{-6} and 10^{-8} . The optimization yields a value of $5.7 \cdot 10^{-7}$ for the sonic conductance of the venting nozzle and $1.9 \cdot 10^{-7}$ for the nozzle controlling the deaeration. The results of measurement and simulation in case of a pressure gain caused by opening the venting nozzle are depicted in 18. In this figure the

model with default values for the sonic conductance and the optimized model are compared with the measurement. It shows that the time dependent behavior is well reproduced by the optimized model except for transient effects in the beginning of the measurement. To represent these effects a more detailed model is necessary.

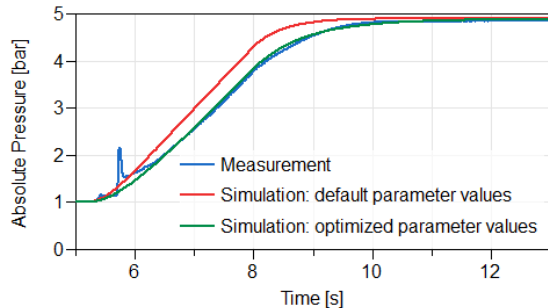


Figure 18. Comparison of measured and simulated C-pressure of distributor valve

4.2 Modeling and simulating a freight train

In another example the brake system of a freight train is modeled and simulated. Freight trains consist of a large number of identical cars connected by coupling elements and a long pneumatic line. Pneumatic effects due to the large volume and the length of the line strongly influence the performance of the system. Their investigation requires large experimental effort which is reduced by using system simulation.

In order to ease the model generation of these trains the library offers a model called "generic train". This model contains a front car connected to a vector of generic car models "railcar i[]". The corresponding connections are implemented by using a for-loop in the equation section and yield a model with a selectable number of car in series.

Figure 19 shows simulation results of the deaeration of the continuous brake pipe (BP) in case of a brake application in a freight train with 50 cars. The brake pipe is the through pneumatic signal line of freight trains which causes the aeration of the brake cylinders with compressed air from a reservoir, if the pressure in the line drops below a particular value¹. Comparing the pressure of the brake pipe in the first car (BP_1) with the last car (BP_50) one can easily observe that the pressure drop in cars in the front is significantly faster than those in the rear part of the train. This dynamic behavior is caused by the limited propagation velocity of the pressure drop as well as the large volume of the long pneumatic line and leads to a strong delay of the pressurization of the brake cylinders at the end of the train. This is shown in Figure 19 by the comparison of the cylinder pressure in first car (C_1) and in the last car (C_50). Due to the delay high longitudinal forces might occur which in turn lead to undesirable

¹This is the principle of an indirect pneumatic brake system as described in (Breuer, 2006) and (Knorr-Bremse, 2003)

oscillations of the train set.

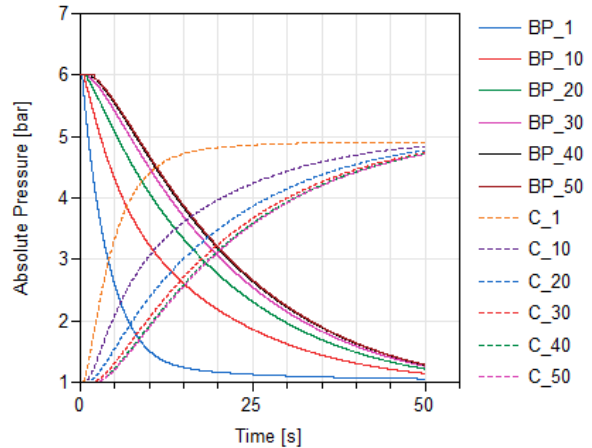


Figure 19. Simulation results of pressure in brake pipes (BP_i) and brake cylinders (C_i) of car $i=1, 10, 20, 30, 40$ and 50 in freight train

5 Conclusion and Outlook

By the application of the library "Virtual Train Brakes" for the analysis of the brake systems of a passenger train and a freight train the potential of system simulation during the design and verification of railway brakes is presented. The possibility to select models of typical technical components and to use generic models for varying system configurations drastically ease the generation of models of railway brakes. Simulations of brake application scenarios generate useful knowledge of the system behavior. This can be used to verify a system against its requirements and to analyze the states of the system yielding the resulting behavior. Furthermore, the application of numerical optimization allows to study the system behavior with respect to sensitive parameters and to identify parameters for the validation of models.

Nevertheless, the development of the library is an ongoing process and the presented results motivate further investigations. The implemented models of the library yet focus on simulations of the pneumatic actuation of frictional brakes. The next step is to add further brake types as listed in Figure 1. This means to develop and integrate models of dynamic brakes and non-adhesion dependent brakes in order to simulate and analyze further brake application modes and blending scenarios. Another expansion of the library is the consideration of vehicle dynamics during brake applications. In the current version of the library the railway vehicle is modeled as a rigid mass on a straight track. This is sufficient to estimate the brake distance. However, to analyze the longitudinal dynamic behavior of the vehicle caused by brake applications more detailed models are necessary. These models might include curved tracks, inclination and downhill slope forces as described in (Spiryagin et al., 2014). Beside the development and integration of more detailed dynamic mod-

els it is also possible to combine brake system models of the presented library with railway vehicle models of other libraries, such as models presented in (Heckmann and Grether, 2017).

Concerning frictional brakes the thermal behavior of brake discs needs to be taken into account during system simulations. The goal is to estimate the temperature of the brake disc during brake applications which influences wear and the friction between disc and pad as described in (Ostermeyer, 2001). This demands to estimate the energy absorbed by the discs and to consider the heat transfer due to conduction, radiation and convection. Additionally, more attention needs to be paid to the modeling of the instantaneous friction coefficient between disc and pad which is influenced by the contacting materials, their temperature, relative speed and normal forces in the contact area. Parameters of models describing these correlations can be estimated from dynamometer measurements as shown in (Lee and Kang, 2015).

Finally, the use of system simulation during the development process demands the validation of entire system models. Therefore, it is necessary to obtain data from train tests, such as integration tests during static and dynamic commissioning, and to perform analog system simulations to verify the system models and ensure that the simulation results are reliable.

Acknowledgments

The presented results arise from an ongoing scientific cooperation of the Institute of System Dynamics and Control of the "German Aerospace Center" and the "Knorr-Bremse Systeme für Schienenfahrzeuge GmbH". The resulting Modelica library is a prototype for an in-house tool and is not supposed to be publicly available. I would like to thank all colleagues from Knorr-Bremse for the strong support during the common project.

References

Jean-Marc André, Hartmut Mann, and Jörg-Johannes Wach. Integrierte Bremssteuerungsmodulare zur schnellen Umsetzung sehr unterschiedlicher und komplexer Anforderungen. *ZEVrail*, 141(10):397–401, October 2017.

Thomas H.F. Anton. *Entwicklungs- und Einföhrungsmethodik für das Projektierungswerkzeug Pneumatiksimulation*. PhD thesis, Technische Universität München, 2010.

Peter Beater. *Pneumatic Drives*. Springer-Verlag Berlin-Heidelberg, 2007. ISBN 10 3-540-69470-6.

Bert Breuer. *Bremsenhandbuch*. ATZ / MTZ-Fachbuch. Vieweg+Teubner Verlag, Wiesbaden, 2006. ISBN 9783663094418.

Commission Regulation (EU) No 1302/2014. Concerning a technical specification for interoperability relating to the 'rolling stock - locomotives and passenger rolling stock' sub-system of the rail system in the European union, November 2014.

Reinhard Haberfellner and Walter F. Daenzer, editors. *Systems Engineering*. Verl. Industrielle Organisation, Zürich, 11th edition, 2002. ISBN 385743998X.

Andreas Heckmann and Gustav Grether. The DLR Railway-Dynamics Library: the Crosswind Stability Problem. In *Proceedings of the 12th International Modelica Conference*, number 132, pages 623–631, 2017.

INCOSE. *Systems Engineering Handbook*. Wiley, 4th edition, 2015.

Chul-Goo Kang, Ho-Yeon Kim, Min-Soo Kim, and Byeong-Choon Goo. Real-time simulations of a railroad brake system using a dspace board. In *2009 ICCAS-SICE*, pages 4073–4078, Aug 2009.

Knorr-Bremse. *Grundlagen der Bremstechnik*. Knorr-Bremse Systeme für Schienenfahrzeuge GmbH, 2003.

Martin Kuhn, Martin Otter, and Loic Raulin. A multi level approach for aircraft electrical systems design. In *Proceedings of the 6th International Modelica Conference, Germany, Bielefeld, March 3-4, 2008*, March 2008.

Nam-Jin Lee and Chul-Goo Kang. The effect of a variable disc pad friction coefficient for the mechanical brake system of a railway vehicle. *PLoS ONE*, 10(8):e0135459, July 2015. ISSN 1932-6203.

DIN EN 14198. Railway applications - Braking - Requirements for the brake system of trains hauled by locomotives, March 2005.

DIN EN 15734-1. Railway applications - Braking systems of high speed trains - Part 1: Requirements and definitions, September 2013.

Modelon AB. *Modeling of Pneumatic Systems - Tutorial for the Pneumatics Library V 1.3/1.5.2*, 2010.

Saurabh Mittal, Umut Durak, and Tuncer Ören. *Guide to Simulation-Based Disciplines: Advancing our Computational Future*. Simulation Foundations, Methods and Applications. Springer, 2017.

G. P. Ostermeyer. Friction and wear of brake systems. *Forschung im Ingenieurwesen*, 66(6):267–272, Oct 2001.

Andreas Pfeiffer. Optimization library for interactive multi-criteria optimization tasks. In *Proceedings of the 9th International Modelica Conference, Munich, Germany, September 03-05, 2012*, pages 669–679, 2012.

L. Pugi, M. Malvezzi, A. Tarasconi, A. Palazzolo, G. Cocci, and M. Violani. HIL simulation of WSP systems on MI-6 test rig. *Vehicle System Dynamics*, 44(sup1):843–852, 2006.

Maksym Spiriyagin, Colin Cole, Y Sun, Mitchell Mcclanachan, Valentyn Spiriyagin, and Tim McSweeney. *Design and Simulation of Rail Vehicles*. 05 2014. ISBN 9781466575660.

Mirko Tischer and Dietmar Widmann. Model based testing and hardware-in-the-loop simulation of embedded canopen control devices. pages 06/19–06/28. iCC, CAN in Automation, 2012.

Drilling Library: A Modelica library for the simulation of well construction

Reza Dadfar¹ Stéphane Velut¹ Per-Ola Larsson¹
Mathias Strandberg¹ Håkan Runvik¹ Johan Windahl¹
Pål Kittilsen² John-Morten Godhavn² Åsmund Hjulstad²

¹Modelon, Sweden, {reza.dadfar, stephane.velut, per-ola.larsson, mathias.strandberg, hakan.runvik, johan.windahl}@modelon.com
²Equinor, Norway, {pkit, jmgo, ahju}@equinor.com

Abstract

A Modelica library for the simulation of well construction (drilling) is presented in this paper. The library contains most of the components of a drilling system. The simulation model is capable to capture the main dynamics of the well, including the hydraulics, the mud transport, the mechanics of the drill string as well as the drill bit interaction with the bore hole. The library is well suited to simulate the well operation and to support the development of new technologies. The modelling assumptions of the library's components are first reviewed. Subsequently, an experiment is performed to test the rotational and translational frictions.

Keywords: Drilling library, fluid-structure interaction, well construction, borehole, surge and swab, mud, Modelica.

1 Introduction

The cost of oil and gas production includes exploration, well and facility construction, operation and cessation. Among all, well construction makes up a considerable part of the cost.

A variety of parties contribute to the well construction, i.e. the oil company, the owner of the drilling rig, the rig-building yard, manufacturers of drilling equipment, and companies that provide equipment and services during the drilling operation. Their relative roles have been refined over years and is not easily changeable. The opportunities from new technologies may require small modifications in the contribution of several parties and is difficult to achieve considering commercially and timewise detachment players. The larger changes that major players can realize may require significant changes to roles and responsibilities (de Wardt, 2014).

However, once benefit is shown, changes are more achievable. In this regard, modeling and simulation can be used as a shared language which may facilitate the cooperation between the actors.

The multitude of simulation tools in the well construction is specialized for developing and validating well designs, and not for developing new technologies. The relevant questions are more of the type: "Is this steel pipe good enough?" as opposed to "what behavior will this system exhibit?". The tools are often discipline specific, vendor proprietary and with few or no extension interfaces for combining results. In addition, there is still a clear disconnection between the simulation tools for downhole processes and topside automation.

Larger changes in the construction methods have the subsequent structural changes in simulation tools. This may be difficult to achieve in the mature commercial tools.

Well construction is typically divided into two phases: drilling and completion. The well is drilled in sections. First a wide hole (13"-30") is drilled a few hundred meters down. The borehole wall is secured by running casing, an open-ended steel cylinder, the same length as the drilled hole. The bottom end, both the annular space outside the casing and inside the casing, is filled with cement. A smaller hole is then drilled through the cement and continuing further towards the target. The steps are repeated until a 3"-9" hole is drilled in the target reservoir, preferably along several hundred meters of hydrocarbon-filled rock, some thousand meters away from the start.

Drilling fluids are pumped through the drill pipe and up to the annulus to transport drilled cuttings out of the hole, to cool down the equipment and to stop formation fluids from uncontrolled flow into the well.

During the completion phase, various equipment is installed to make the well ready for production. The type of completion equipment varies significantly, from a very simple open hole to wells equipped with sensors, pumps, flow control devices, chemical injection lines, gas lift, etc.

The modeling of the well construction involves a lot of challenges. The modelling domains (flow circulation, rotation and translation of drill pipe) are physically coupled both along the drill-string and near the bit. The thermal energy domain contains convection-driven flow (along the flow path), radial diffusion (between drill string and surrounding rock), and variable fluid composition. The fluid is non-Newtonian and the rheological properties depends on time and temperature. We should not try to always deal with all these effects simultaneously. For many problems, it is more appropriate to select relevant physical phenomena and system components. This is where Modelica may prove useful (Fritzson, 2014).

In the following, some industrial applications are listed as examples of new technologies or components where understanding of the dynamics of physical processes is important for both overall system and control system design. The development includes significant modelling and simulation work. In some cases, it may have been necessary to develop simulation models from scratch. A language like Modelica and a shared base of libraries may reduce the simulation costs, decrease the threshold to use dynamic simulation (i.e. in research, concept development, engineering and testing), and may make it easier to include custom components in simulated demonstration, operations preparations or training.

The following examples are not indication of Modelica use (known to the authors), but rather recent technology developments of a kind that would benefit from a general and flexible drilling library.

1. Maintaining downhole pressure within desired limits during drilling operations is a well-studied problem. Industrial solutions exist and is referred to as Managed Pressure Drilling (Chin, 2012). The following are some examples:

a) On Norwegian continental shelf, the first offshore applications are described in (Bjørkevoll et. al., 2008), (Eck-Olsen et. al., 2005), where rich models are used as input to automatic choke control. Kaasa et.al, (Kaasa et. al., 2012) argue for using a simplified process model in design of estimation and control algorithms and for reducing the complexity and footprint of the control system software. Both approaches still use a top-side choke for pressure control, and neither solve issues that arise when drilling from floating installations; heave-induced downhole pressure oscillations.

b) Another approach is to use a downhole valve to reduce the pressure oscillations (Kvernland, 2018).

c) For deep water, a reduced liquid level in the drilling riser (as opposed to increased backpressure topside) is used to control downhole pressure. Various alternatives

exist (Godhavn, 2014) with subsea pumps, in combination with sealing devices and valves.

d) Conventionally, the fluid flows downwards inside the drill pipe, returning to the annular outside space. An idea is to use a drill pipe with two concentric channels, with return flow inside the pipe, (Aleksandersen et. al., 2015).

2. A similar application is automatic well control solutions, where the task is to first detect a kick/influx, then stop the influx and finally handle the influx by circulating it out using well control equipment. The flow in the annulus is multiphase and contains drilling fluid, rock particles and possibly gas from formation influx.

3. An additional application example that would benefit from a flexible drilling library are drill-string vibration (detection and handling).

4. Automatic mud mixing is another example. Drilling mud is designed with several properties, where density and viscosity are the most important properties. Maintaining and controlling these properties are very important. Simulating the mud properties through the topside process as well as the drill pipe and the annulus may be a challenge as the properties are changing with time, pressure and temperature.

These are examples of new types of equipment and changes in model structure that are not easily implementable in typical simulation software when conceiving the idea. The drilling library presented in the current paper is the first version of that versatile tool that aims at supporting the development of new technologies for drilling processes. Simulation models could ultimately be used as knowledge carriers between the parties involved in the well construction or as digital twin to monitor the well operation.

2 Drilling Library

The simulation of the drilling operation using a flexible tool makes it possible to obtain a realistic behavior of the operation, graphical representation of the results, sensitivity analyses and control design. In the modern, high pressure and high temperature, HPHT, wells, it is crucial to have a profound understanding of the dynamic behavior of the well for accurate well planning, training and operational assistance. In this regard, Modelica is a well-known tool providing a flexible platform to simulate multi domain physics including the thermo-flow and mechanical dynamics. Hence it is well suited for the simulation of the drilling operation and control.

The aim is to build a library to simulate:

1. Well hydraulics: From the main pump at the surface to the topside interface.
2. Drill string mechanics:
 - Detailed mechanics of the string (torsion and elongation)
 - Rotational friction between string and surrounding fluids
 - Interaction of the drill bit with the surroundings to describe the bore hole growth
3. Mud transport: The transported fluid that is normally a mixture of oil, water and granular particles.

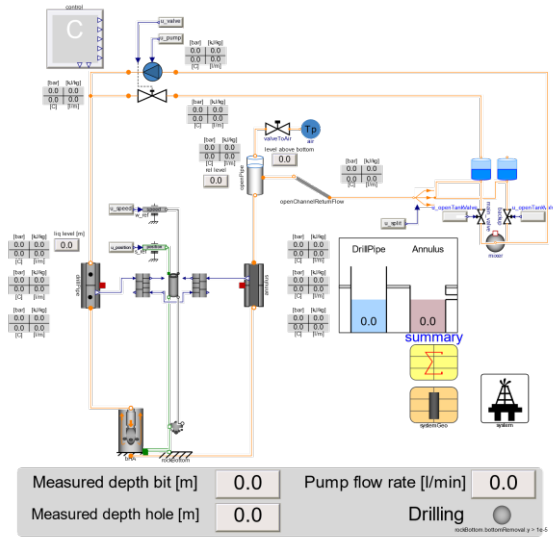


Figure 1. Modelica model of a drilling system

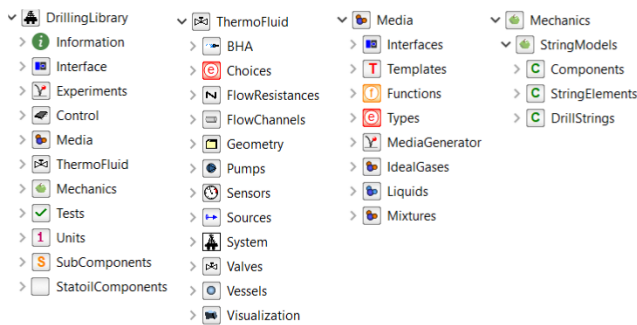


Figure 2. Overview of the Modelica drilling library

The current library illustrated in Figure 1 and in Figure 2 is organized in packages;

1. Media package: for the state equations and transport properties that defines the physical properties of the drilling fluid *i.e.* mud.
2. Thermal package for the thermofluid components used in the hydraulic and thermal modeling of the drilling operation.

3. Mechanics package for the mechanical parts of the system including the drill string and the mechanical friction correlations.
4. Control package: contains all the control blocks required to operate the entire drilling operations.
5. Experiments package: contains the system models, where different scenarios using mechanical and thermo-fluid components can be built based on a template.

2.1 Media package

In this package the medium properties of the drilling fluid are implemented. The medium properties depend on temperature T , pressure p and mass fraction X ;

$$\rho = \rho(p, T, X) \quad (1)$$

$$h = h(p, T, X) \quad (2)$$

$$\mu = \mu(p, T, X) \quad (3)$$

where ρ, h, μ are density, specific enthalpy and dynamic viscosity, respectively.

The drilling fluid is made of five components, namely

- Two solid constituents: low and high gravity granular particles
- Two liquid constituents: base oil and brine
- One gas.

The gas is treated as an ideal gas; the properties of the solid particles are constant; and the liquids are governed by a bilinear equation of state:

$$d\rho_k = \left(\frac{\partial \rho_k}{\partial p}\right)_T dp + \left(\frac{\partial \rho_k}{\partial T}\right)_p dT + \left(\frac{\partial^2 \rho_k}{\partial p \partial T}\right)_{pT} dp dT \quad (4)$$

$$\left(\frac{\partial \rho_k}{\partial T}\right)_p = -\rho_{0,k} \alpha_k \quad (5)$$

$$\left(\frac{\partial \rho_k}{\partial p}\right)_T = \frac{\rho_{0,k}}{\beta_k} \quad (6)$$

where k =base oil and brine, respectively.

The specific enthalpy of the liquids was assumed to depend only on temperature and composition, *i.e.* $h = h(T, X)$. The implementation can be easily extended to include pressure dependency $h = h(p, T, X)$. The mud mixture properties are finally obtained as

$$1/\rho = \sum_{i=1}^n X_i/\rho_i \quad (7)$$

$$h = h_{ref} + \sum_{i=1}^n c_{p,i} X_i (T - T_{ref}) \quad (8)$$

$$\mu = \mu_0 10^{\frac{\mu_1}{T-T_0}} \quad (9)$$

where $c_{p,i}, \mu_0, \mu_1, T_0, h_{ref}, T_{ref}$ are constants, and $i = 1, \dots, n$; where n is the number of components.

2.2 Thermal Package

This package contains a hierarchical component structure used in the thermo-fluid subsystems. This domain communicates with the mechanical domain through appropriate interface (see 2.3).

2.2.1 Flow channels

The flow channel describes the flow through the drill string and the annulus (Figure 3 and Figure 4). The component model is implemented by combining two approaches:

The first approach is a finite volume discretization on a staggered grid where the mass and energy balance are implemented in a dynamic volume and the momentum equation is modeled in a flow resistance/static head component. This component is keeping track in a dynamic fashion of the pressure, the enthalpy and the composition in the bore hole.

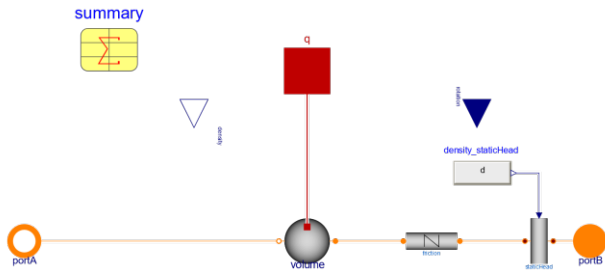


Figure 3. Modelica model of a pipe segment, implemented as the serial connection of a control volume, a friction loss model and a static head computation

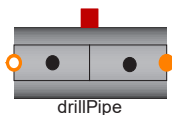


Figure 4. Drill pipe component implemented as a series of pipe segments

The balance equations read:

$$\frac{\partial h}{\partial t} + \left(\frac{\dot{m}}{\rho A}\right) \frac{\partial h}{\partial x} - \frac{1}{\rho} \frac{dp}{dt} - \frac{P}{\rho A} q'' = 0 \quad (10)$$

$$\frac{\partial p}{\partial x} + \rho g \frac{\partial z}{\partial x} + \frac{\partial \tau_w}{\partial x} = 0 \quad (11)$$

$$A \frac{\partial \rho}{\partial t} + \frac{\partial \dot{m}}{\partial x} = 0 \quad (12)$$

$$A \frac{\partial \rho X_i}{\partial t} + \frac{\partial \dot{m} X_i}{\partial x} = 0 \quad (13)$$

where X_i is the mass fraction for each component, $i = 1, \dots, n$, and \dot{m} , τ_w , A , P , q'' are mass flow rate, wall shear stress, pipe cross section area, pipe wetted perimeter and heat flux, respectively. In the momentum equations, the unsteady and convection terms, and in the energy equation, the conduction and the viscous dissipation terms are not considered.

The Modelica built-in operator “spatial distribution” is used in parallel with the finite volume channel to model the transport of the solid particles without any numerical diffusion and to increase the robustness and the speed of the simulation. This operator approximates the solution of the plug-flow partial differential equation in a robust way. The operator supports reverse flow and keeps track of the spatial distribution of the quantity, when the flow varies via sampling, interpolation and shifting of the previous distribution (Fritzson, 2014). The species balance can then be re-written in a form that is suitable for the spatial distribution operator:

$$\frac{dX_i}{dt} + u \frac{dX_i}{dx} = 0 \quad (14)$$

$$X_i(0, t) = X_{i,A} \quad (15)$$

$$X_i(L, t) = X_{i,B} \quad (16)$$

The species’ mass fraction returned by the spatial distribution operator is passed to the control volume to be used in the species balances therein. The velocity that is sent to the spatial distribution operator is computed as follows;

$$u(t) = \frac{\dot{m}_A + \dot{m}_B}{2\rho}, \quad (17)$$

where A and B are the boundaries of the control volume.

2.2.2 Flow resistances

The pressure loss has to be estimated for each component in the mud circulation system. To obtain the pressure loss in the drill pipe and annulus, wall friction is estimated as

$$\frac{\partial \tau_w}{\partial x} = \frac{2f}{\rho D_h A^2} \dot{m} |\dot{m}|, \quad (18)$$

where f is the fanning friction factor and D_h is the hydraulic diameter of the pipe. The friction factor f is determined, depending on the selected friction correlation. For this library two correlations are provided, i.e. Herschel-Bulkley and Power-Law (Zamora et. al. , 2002) and (Zamora et. al. 2005).

In the drill bit and bottom hole assembly, the pressure loss is determined as

$$\Delta p_{bit} = \frac{1}{2\rho(c_v A)^2} \dot{m} |\dot{m}| \quad (19)$$

$$\Delta p_{bha} = \frac{c_{dt}}{\rho_{dt}} \rho^{0.86} \dot{m} |\dot{m}|^{0.86} \quad (20)$$

where c_v , c_{dt} , ρ_{dt} are the discharge coefficient, pressure coefficient for down-hole tool and reference fluid density, respectively.

2.2.3 Bottom hole assembly

The length of the wellbore increases during drilling. This effect is modelled in the fluid system by a variable volume component located between the drill bit and

bottom hole assembly. The volume is estimated based on the drilling string displacement. The interaction is modeled by two flange ports as shown in Figure 5. As a part of fluid-mechanic interaction, the fluid system introduces an equivalent force on the drill string;

$$F = p_{bha} A_{bit} \quad (21)$$

The fluid system provides also the information of densities in the drill string and annulus and hand it over to mechanics to determine the buoyancy force.

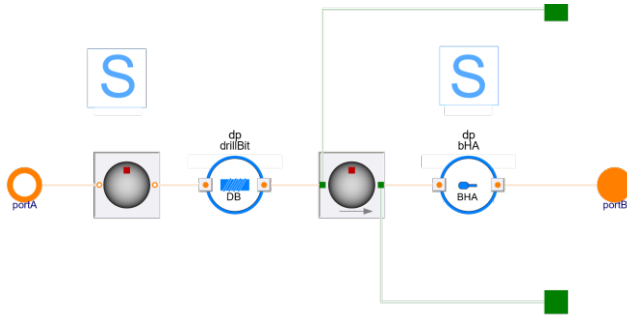


Figure 5. Modelica model of the bottom hole assembly and drill bit.

2.2.4 Vessels

Two open tanks are considered in this system. Each tank contains different mud compositions. Hence the density of the circulating mud can be controlled by using mud from each or the combination of two tanks.

In the mud circulating system, the top of annulus opens to the atmosphere. To model this, “open pipe” is added to the top of the annulus. This model is connected to the return line through a third fluid port. The free surface level is estimated, and the result is visualized graphically.

2.3 Mechanics Package

The Mechanics package includes models covering the drill string and its mechanical interaction with the well bottom and the annulus. The drill string also interfaces to the fluid. The string is assumed rotating only around its own axis and does not include eccentricity. It’s also assumed hanging free in completely vertical parts of the well and resting on the bottom of the well in parts with inclination.

2.3.1 Drill-string

The drill string was implemented using a lumped mass approach, where each segment of the string has 2 degrees of freedom, one translational and one rotational, defined along the well trajectory. The drill string is assembled by a number of these elements and their interactions.

2.3.2 Elements

Each element consists of two masses connected with one spring damper element. The masses are calculated from geometric input, the inner radius (r_{inner}), the outer

radius (r_{outer}), Length, ρ , Young's modulus (E), shear modulus (G). The spring stiffness is calculated so that with a given force or torque, the static deformation of the element is correct. The elements also contain a damping factor, that is currently overestimated, which leads to more damping of rotational and translational oscillations. More work is needed to estimate effects and improve the damping effects in the system.

2.3.3 Buoyancy and gravity

Each element has access to its current position along the well trajectory and can acquire information about the local fluid densities inside the string and annulus, and the well inclination. With this information the correct buoyancy effect can be calculated. The buoyancy has an effect on the normal force between the string and annulus. This will also calculate the gravitational effect along the string.

2.3.4 Friction

The friction applied in the contact is a 2d model defining a force ellipse for break-away friction. This enables the study of the connected effects of rotational and translational friction. Typical example is when the string is twisted but at rest in the well, and the string is pulled out, the friction is saturated by the translational motion and at one point the rotational friction cannot hold the twisted string and unwinds.

The friction model is parameterized using a table providing Coulomb and viscous friction coefficients as function of temperature.

2.3.5 Bottom removal

The bottom removal model is designed to remove mass and increase well depth of the system. It consists of a contact model, initialized at the defined start depth of the well, and a removal model. The removal model defines a relationship between bit rotational speed, contact force and gives the penetration speed. This speed is then applied to the bottom contact and moves it further down along the predefined well trajectory data.

2.3.6 Experiments

Several Modelica experiments have been implemented using reconfigurable system templates provided in the library. The aim was to assess the behavior of the drilling system and of the library components under various conditions with respect to both the mechanical and the fluid domains. Experiments in the fluid domain included switching on and off the mud pump and tracking the pressure and flow along the string and through the bit; adding solid particles and tracking the front propagation; surge and swab scenarios. As far as the mechanical sub-system is concerned, both rotational and translational friction tests were conducted. The latter is explained in more details in this section.

During drilling long wells, especially with near-horizontal trajectories, it is important to monitor

translational and rotational friction between drillstring and borehole. Deviation from expected behavior causes difficulties under development (Johancsik et. al., 1984). It can lead to poor hole-cleaning performance (drilled rock is not being transported out) and well path tortuosity (Skillingstad et. al., 2000). Automatic execution of friction tests has recently been applied to offshore operations. Standardization of the test procedure and using software to compensate for variations in test execution are important for accurate results (Cayeux et. al., 2017).

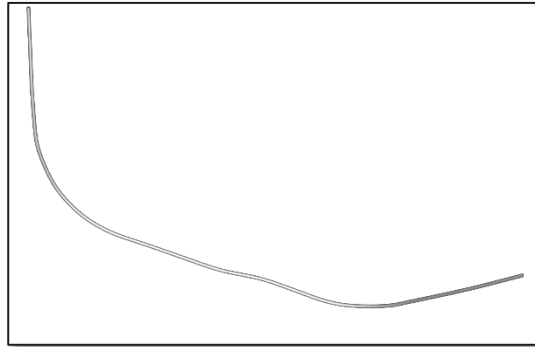


Figure 6. Three-dimensional well trajectory

The trajectory of the well is shown in Figure 6. The geometrical data of the well are propagated to both the thermo-fluid and mechanics sub systems.

The test is performed by controlling the movement of the top block (see Figure 7). The drilling operation is performed up to $t=80$ s. Then, the block stops its vertical descent and let the drill-bit to drill off the bottom.

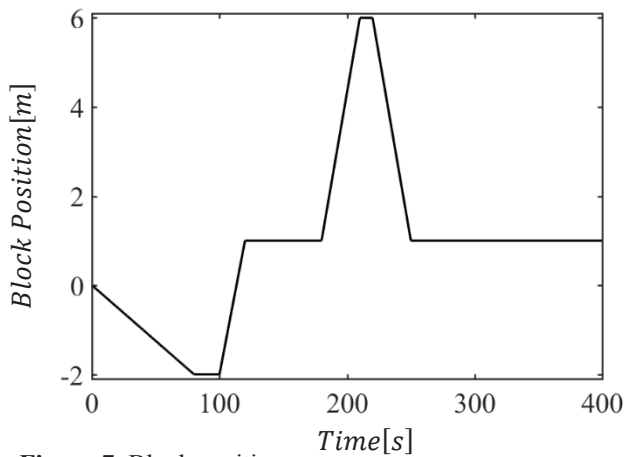


Figure 7. Block position

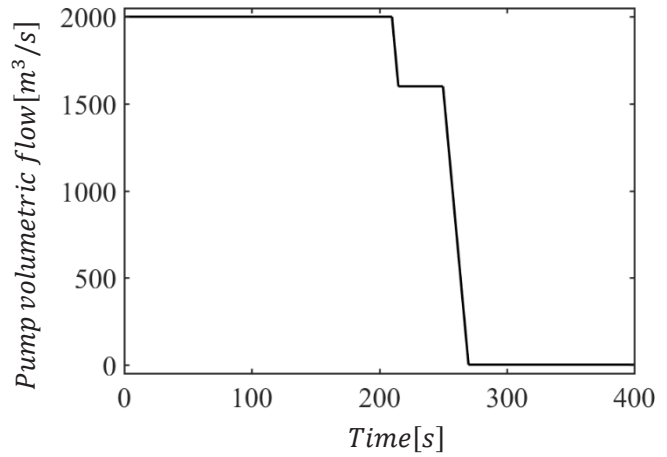


Figure 8. pump volumetric flow

At $t=100$ s, the string is pulled up 3m for 20s, where the top block stops the rotation of the drill string. Then, the string is pulled up and pushed down for 5m to measure the hook force and to estimate the friction along the well. The friction estimation was not actually performed in this example.

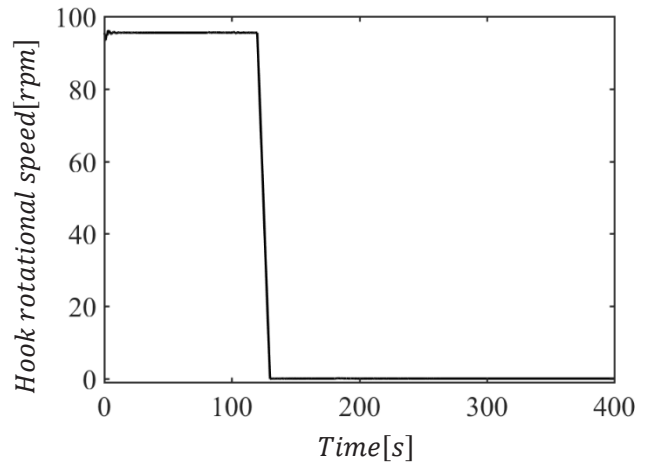


Figure 9. Hook rotational speed

Then, the string is pulled up and pushed down for 5m to measure the hook force and to estimate the friction along the well. The friction estimation was not actually performed in this example. Block position, pump volumetric flow and the hook rotational speed are shown in Figure 7, Figure 8 and Figure 9.

Initial transients in the bit rotational speed is shown in Figure 10. The transient is because the string starts with zero twist, and the initial rotations starts to twist the string, successively overcoming the friction force and then the bit starts to rotate.

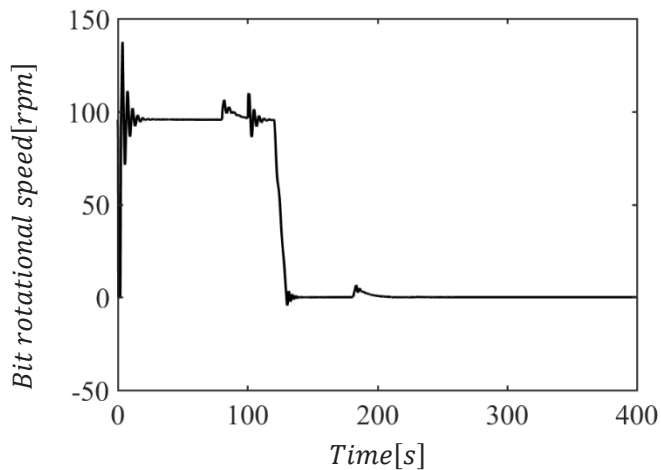


Figure 10. Bit rotational speed

Contact force in the bottom is shown in Figure 11. The force decays at $t=80s$ then the bit drills off the bottom.

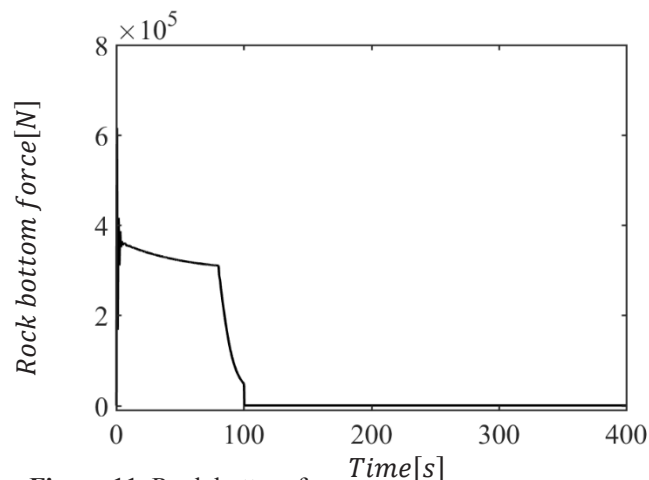


Figure 11. Rock bottom force

The difference in bit and hook torque are shown in Figure 12. When the bit is off the bottom, the reaction torques decreases, and the bit torque is reduced to zero. The drill string is still twisted and stuck because of the friction. This reaction torque can be seen in the hook torque.

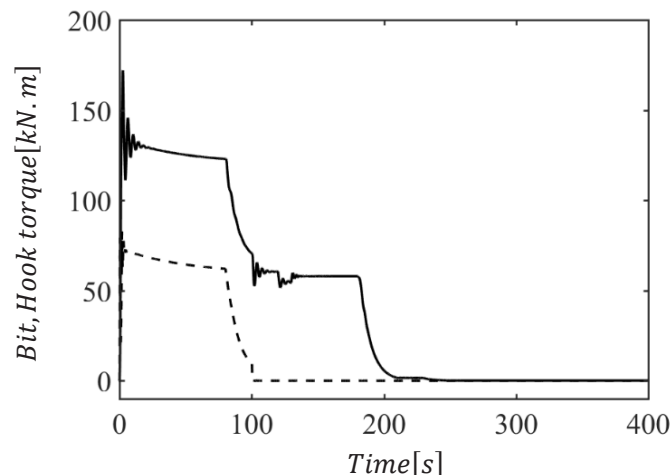


Figure 12. Bit and Hook torque; the solid line represent the hook torque while the dotted line shows the bit torque

When the string is pulled up at $t=180s$, the friction is saturated because of the axial translational movement and it can no longer resist the twist and unwinds.

3 Conclusions and outlook

The drilling library presented in the current paper is a first step towards a versatile tool for new technology development for well construction. It shows the potential of the Modelica technology in a simple but quite complete drilling system with fluid-mechanical interactions. The chosen implementation strategies show computation times that are suitable for interactive work. In Equinor, Modelica is used for control studies and design within oil and gas production and processing, and also as a tool to produce FMUs for integration in process simulators. The modularity and extensibility of Modelica is expected to lower the threshold for using dynamic simulation in prototyping systems and control functions for well construction. The proposed library is currently planned to be used by Equinor within research work and in cooperation with academia. The authors welcome also industrial partners to participate in the further development of the library.

4 References

- Aleksandersen, J., & Vestavik, O. M. (2015). Dual-Drillpipe Method Shows Success in PMCD Wells With Cuttings Return. *Journal of Petroleum Technology*, 67, 32-35. doi:10.2118/0415-0032-JPT
- Bjørkevold, K. S., Molde, D. O., Rommetveit, R., & Sylto, S. (2008). MPD operation solved drilling challenges in a severely depleted HP/HT reservoir. *IADC/SPE Drilling Conference*. doi:10.2118/112739-MS
- Cayeux, E., Skadsem, H. J., Daireaux, B., & Holand, R. (2017). Challenges and solutions to the correct interpretation of drilling friction tests. *SPE/IADC Drilling Conference and Exhibition*. doi:10.2118/184657-MS
- Chin, W. C. (2012). *Managed pressure drilling: modeling, strategy and planning*. Gulf Professional Publishing.
- de Wardt, J. (2014). The drilling business model: driver or inhibitor of performance and innovation. *IADC/SPE Drilling Conference and Exhibition*. doi:https://doi.org/10.2118/167933-MS
- Eck-Olsen, J., Pettersen, P.-J., Ronneberg, A., Bjørkevold, K. S., & Rommetveit, R. (2005). Managing pressures during underbalanced cementing by chok ing the return flow; innovative design and operational modeling as well as operational lessons. *SPE/IADC Drilling Conference*. doi:10.2118/92568-MS
- Fritzson, P. (2014). *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. John Wiley & Sons.
- Gerogiorgis, D. I. (2006). Dynamic oil and gas production optimization via explicit reservoir simulation. *Computer*

- Aided Chemical Engineering*, 21, 179-184.
doi:10.1016/S1570-7946(06)80043-X
- Godhavn, J.-M., Hauge, E., Molde, D. O., Kjøsnes, I., Gaassand, S., Fossli, S. B., & Stave, R. (2014). ECD Management Toolbox for Floating Drilling Units. *Offshore Technology Conference*. doi:10.4043/25292-MS
- Johancsik, C. A., Friesen, D. B., & Dawson, R. (1984). Torque and drag in directional wells-prediction and measurement. *Journal of Petroleum Technology*, 36, 987-992. doi:10.2118/11380-PA
- Kaasa, G.-O., Stamnes, Ø. N., Aamo, O. M., & Imsland, L. S. (2012). Simplified hydraulics model used for intelligent estimation of downhole pressure for a managed-pressure-drilling control system. *SPE Drilling & Completion*, 27, 127-138. doi:10.2118/143097-PA
- Kvernland, M., Christensen, M. Ø., Borgen, H., Godhavn, J.-M., Aamo, O. M., & Sangesland, S. (2018). Attenuating Heave-Induced Pressure Oscillations using Automated Down-hole Choking. *IADC/SPE Drilling Conference and Exhibition*. doi:10.2118/189657-MS
- Skillingstad, T. (2000). At-bit inclination measurements improves directional drilling efficiency and control. *IADC/SPE Drilling Conference*. doi:10.2118/59194-MS
- Zamora, M., & Power, D. (2002). Making a case for AADE hydraulics and the unified rheological model. *AADE 2002 Technology Conference Drilling & Completion Fluids and Waste Management, Houston, USA*.
- Zamora, M., Roy, S., & Slater, K. (2005). Comparing a basic set of drilling fluid pressure-loss relationships to flow-loop and field data. *AADE 2005 National Technical Conference and Exhibition*, (ss. 5-7).

