PROCEEDINGS

14th Modelica Conference

September 20-24, 2021
Linköping, Sweden

The 14th International Modelica Conference is sponsored by:

DASSAULT SYSTEMES
Julia computing
LTX
CLAYTEX
Maplesoft
Modelon
TLK-Thermo GmbH
TOSHIBA
WOLFRAM SYSTEMMODELER

The 14th International Modelica Conference is organized by:

LINKÖPING UNIVERSITY
RISE Research Institutes of Sweden
Modelica Association

Proceedings of 14th Modelica Conference 2021
Linköping, Sweden, September 20-24, 2021

Editors:
Martin Sjölund, Linköping University
Lena Buffoni, Linköping University
Adrian Pop, Linköping University
Lennart Ochel, RISE Research Institutes of Sweden AB

Published by:
Modelica Association and Linköping University Electronic Press

ISBN:	978-91-7929-027-6
Series:	Linköping Electronic Conference Proceedings, No. 181
ISSN:	1650-3686
eISSN:	1650-3740
DOI:	https://doi.org/10.3384/ecp21181

Organized by:
Linköping University
SE-581 83 Linköping
Sweden

In cooperation with:
Modelica Association
c/o PELAB, Linköpings University
SE-581 83 Linköping
Sweden

Conference location:
Virtual conference
https://2021.international.conference.modelica.org/

Front cover photo
Used with permission: Fair at the piazza outside Linköping Konsert & Kongress.
Copyright 2015 Visit Linköping & Co. https://visitlinkoping.mediaflowportal.com/

Preface

The Modelica Conference is the main event for users, library developers, tool vendors and language designers to share their knowledge and learn about the latest scientific and industrial progress related to Modelica, the Functional Mockup Interface (FMI), System Structure & Parametrisation (SSP), Distributed Co-Simulation protocol (DCP) and eFMI.

Since the start of the collaborative design work for Modelica in 1996, Modelica has matured from an idea among a small number of dedicated enthusiasts to a widely accepted standard language for the modeling and simulation of cyber-physical systems. The Modelica language was standardized by the non-profit organization Modelica Association which enabled Modelica models to be portable between a growing number of tools. Modelica is the language of choice for model-based systems engineering and is now used in many industries including automotive, energy and process, aerospace, and industrial equipment.

The Modelica Association has since grown to include several projects supporting modeling and simulation, creating a family of inter-related standards complementing each-other. FMI is an open standard that defines a container and an interface to exchange dynamic models using a single file (an FMU). SSP is a tool-independent standard to define complete systems consisting of one or more FMUs including its parameterization that can be transferred between simulation tools. DCP is a platform and standard for the integration of models or real-time systems into simulation environments. eFMI tooling enables the automatic transformation of higher-level acausal model representations (such as Modelica) to causal solutions suitable for integration in embedded systems.

Highlights of the conference include:

- 69 oral presentations, 4 libraries for the Modelica Library Award
- 2 Keynotes
- 5 Tutorials and 2 Industrial User Presentations Sessions
- 9 Vendor Sessions and 9 Sponsors

Welcome

We would have loved to Welcome you in Linköping at the Concert and Congress hall pictured on the front page, but unfortunately even after delaying the conference by 6 months, we have made the decision to move the conference online due to the pandemic. We decided to turn this into an opportunity, by waiving registration fees so that more people would get the opportunity to get involved in the community. We are grateful to our sponsors and the Modelica Association for making this possible.

Another change we have made this year is to extend the conference over the whole week, and we have dedicated half of each day to scientific papers and half to tutorials, vendor sessions and industrial user sessions. Two half-days are dedicated to Industrial User Presentation Sessions, that are not part of the proceedings, but an opportunity for actual users to exchange their experience with the community. This year we have received a total of 81 submissions, all throughly reviewed by our program committee.

Lena Buffoni

Martin Sjölund

Lennart Ochel

Adrian Pop

Modelica News

In the name of the Modelica Association that is co-organizing this event, I also would like to welcome you. It is now already the 14th conference on the Modelica Language, the Functional Mockup Interface and related technology.

Since the number of projects and standards of the Modelica Association is growing, we will give a short overview about the current status in the opening session on Monday afternoon (13:30 – 14:15 CEST) under the traditional heading "Modelica Association News".

Prof. Martin Otter
DLR, Oberpfaffenhofen, Germany
Chairperson of Modelica Association

Keynote Speakers

Dr. Viral Shah
Co-founder and CEO
Julia Computing

Dr. Chris Rackauckas
Director of Modeling and Simulation
Julia Computing

Dr. Chris Laughman
Principal Research Scientist
Mitsubishi Electric Research Laboratories

Dr. Michael Wetter
Berkeley Lab
Energy Technologies Area
Building Technology & Urban Systems Division

New Horizons in Modeling and Simulation with Julia

As modeling has become more ubiquitous, our models keep growing. The time to build models, verify their behavior, and simulate them is increasing exponentially as we seek more precise predictions. How will our tools change to accommodate the future?

Julia's language design has led to new opportunities. The combination of multiple dispatch, staged compilation, and Julia's composable libraries have made it possible to build a next generation symbolic-numeric framework. Julia's abstract interpretation framework enables capabilities such as automatic differentiation, automatic surrogate generation, symbolic tracing, uncertainty propagation, and automatic parallelism. These features have allowed us to build various applications in pharmaceuticals, aerospace, energy, materials, circuits, and much more - demonstrating performance that is many orders of magnitude better.

How can the Modelica community support the transition to decarbonized, grid-flexible buildings?

Due to climate change, the energy sector is undergoing a rapid, fundamental transition. The building sector contributes more CO2 emissions than transportation or industry. To mitigate climate change, building energy systems need to become increasingly electrified and contribute to integration of renewables and storage at scale, while being resilient in view of extreme weather events.

This will lead to new system architectures and controls, and new design flows that can manage the increased complexity. After laying out these challenges and their implications, we will present recent progress on new generation computational tools for building and district energy and control systems design, deployment and operation. The presentation closes with needs that the Modelica community can address to better and faster support the decarbonization of the building sector.

Program Committee

Conference Chairs

Dr. Lena Buffoni, Linköping University, Linköping, Sweden
Dr. Lennart Ochel, RISE AB, Linköping, Sweden

Program Chairs

Dr. Martin Sjölund, Linköping University, Linköping, Sweden
Dr. Adrian Pop, Linköping University, Linköping, Sweden

Conference Board

Dr. Lena Buffoni, Linköping University, Linköping, Sweden
Dr. Lennart Ochel, RISE AB, Linköping, Sweden
Dr. Martin Sjölund, Linköping University, Linköping, Sweden
Dr. Adrian Pop, Linköping University, Linköping, Sweden
Prof. Francesco Casella, Politecnico di Milano, Milano, Italy
Dr. Hilding Elmqvist, Mogram, Lund, Sweden
Prof. Peter Fritzson, Linköping University, Linköping, Sweden
Prof. Martin Otter, DLR, Oberpfaffenhofen, Germany
Dr. Michael Tiller, Ricardo, Inc., Michigan, USA

Program Committee

Prof. Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany
Dr. John Batteh, Modelon, Ann Arbor, Michigan, USA
Dr. Albert Benveniste, INRIA, Rennes, France
Christian Bertsch, Robert Bosch GmbH, Stuttgart, Germany
Volker Beuter, VI-grade GmbH, Darmstadt, Germany
Thomas Beutlich, Germany
Torsten Blochwitz, ESI ITI GmbH
Dr. Scott Bortoff, Mitsubishi Electric Research Laboratories, Cambridge, Massachusetts, USA
Dr. Timothy Bourke, INRIA, Paris, France
Daniel Bouskela, Électricité de France, Paris, France
Dr. Robert Braun, Linköping University, Linköping, Sweden
Dr. Lena Buffoni, Linköping University, Linköping, Sweden
Felix Bünning, Empa/ETH Zürich, Zürich, Switzerland
Prof. Francesco Casella, Politecnico di Milano, Milano, Italy
Prof. Massimo Ceraolo, University of Pisa, Pisa, Italy
Dr. Yan Chen, Pacific Northwest National Lab. Portland, Oregon, USA
Dr. Massimo Cimmino, École Polytechnique de Montréal, Quebec, Canada
Christoph Clauss, Dresden, Germany
Dr. Johan de Kleer, Palo Alto Research Center, Inc., Palo Alto, California, USA
Dr. Hilding Elmqvist, Mogram AB, Lund, Sweden
Dr. Atiyah Elsheikh, Mathemodica.com
Dr. Olaf Enge-Rosenblatt, Fraunhofer IIS EAS, Dresden, Germany
Prof. Gianni Ferretti, Politecnico di Milano, Milano, Italy
Prof. Peter Fritzson, Linköping University, Linköping, Sweden
Leo Gall, LTX Simulation GmbH, München, Germany
Dr. Virginie Galtier, CentraleSupélec, Paris, France
Prof. Anton Haumer, OTH Regensburg, Regensburg, Germany
Dr. Dan Henriksson, Dassault Systèmes AB, Lund, Sweden
Dr. Yutaka Hirano, Woven Planet Holdings, Inc., Tokyo, Japan
Dr. Andreas Junghanns, Synopsys, Inc., Berlin, Germany

Jochen Köhler, ZF Friedrichshafen AG, Friedrichshafen, Germany
Dr. Christian Kral, Electric Machines, Drives and Systems
Dr. Imke Krüger, Dassault Systemes Deutschland GmbH
Dr. Christopher Laughman, Mitsubishi Electric Research Laboratories, Cambridge, Massachusetts, USA
Dr. Mareike Leimeister, Fraunhofer Institute for Wind Energy Systems IWES, Bremerhaven, Germany
Kilian Link, Siemens Energy AG, München, Germany
Dr. Alessandro Maccarini, Aalborg University, Aalborg, Denmark
Kristin Majetta, Fraunhofer IIS EAS, Dresden, Germany
Dr. Marek Matejak, Charles University, Prague, Czech Republic
Dr. Alexandra Mehlhase, Arizona State University, Arizona, USA
Dr. Ramine Nikoukhah, Altair Engineering, Paris, France
Dr. Henrik Nilsson, University of Nottingham, Nottingham, UK
Prof. Thierry S Nouidui, The United African University of Tanzania, Dar es Salaam, Tanzania
Prof. Christoph Nytsch-Geusen, Universität der Künste Berlin, Berlin, Germany
Dr. Lennart Ochel, RISE AB, Linköping, Sweden
Dr. Hans Olsson, Dassault Systèmes AB, Lund, Sweden
Prof. Martin Otter, DLR, Oberpfaffenhofen, Germany
Dr. Kaustubh Phalak, Ingersoll Rand, Davidson, North Carolina, USA
Dr. Adrian Pop, Linköping University, Linköping, Sweden
Johan Rhodin, ModSimTech, LLC, St. Louis, Missouri, USA
Dr. Lisa Rivalin, Facebook, Menlo Park, California, USA
Clemens Schlegel, Schlegel Simulation GmbH, Freising, Germany
Prof. Gerhard Schmitz, Hamburg University of Technology, Hamburg, Germany
Dr. Peter Schneider, Fraunhofer IIS EAS, Dresden, Germany
Prof. Stefan-Alexander Schneider, University of Applied Sciences Kempten, Kempten, Germany
Dr. Gerald Schweiger, TU Graz, Graz,Austria
Dr. Elena Shmoylova, Maplesoft, Waterloo, Ontario, Canada
Dr. Michael Sielemann, Modelon, München, Germany
Dr. Giorgio Simonini, Électricité de France, Paris, France
Dr. Martin Sjölund, Linköping University, Linköping, Sweden
Dr. Ed Tate, Dassault Systèmes, Grand Blanc, Michigan, USA
Dr. Wilhelm Tegethoff, TLK-Thermo GmbH, Braunschweig, Germany
Dr. Bernhard Thiele, DLR, Oberpfaffenhofen, Germany
Dr. Matthis Thorade, Modelon, Germany
Dr. Michael Tiller, Ricardo, Inc., Michigan, USA
Dr. Jakub Tobolar, DLR, Oberpfaffenhofen, Germany
Dr. Hubertus Tummescheit, Modelon, Hartford, Connecticut, USA
Dr. Alfonso Urquia, Universidad Nacional de Educación a Distancia, Madrid, Spain
Dr. Luigi Vanfretti, Rensselaer Polytechnic Institute, Troy, New York, USA
Volker Waurich, TU Dresden, Dresden, Germany
Dr. Michael Wetter, Lawrence Berkeley National Laboratory, Berkeley, California, USA
Dietmar Winkler, University of South-Eastern Norway
Stefan Wischhusen, XRG Simulation GmbH, Hamburg, Germany
Dr. Dirk Zimmer, DLR, Oberpfaffenhofen, Germany
Philipp Zofer, LTX Simulation GmbH, München, Germany
Dr. Wangda Zuo, University of Colorado Boulder, Boulder, Colorado, USA
Dr. Johan Åkesson, Modelon, Lund, Sweden

Contents

Author Index

The Functional Mock-up Interface 3.0 - New Features Enabling New Applications

Andreas Junghanns[1] Torsten Blochwitz[2] Christian Bertsch[3] Torsten Sommer[4]
Karl Wernersson[5] Andreas Pillekeit[6] Irina Zacharias[6] Matthias Blesken[6] Pierre R. Mai[7]
Klaus Schuch[8] Christian Schulze[9] Cláudio Gomes[10] Masoud Najafi[11]

[1]Synopsys, Germany, `Andreas.Junghanns@synopys.com`
[2]ESI ITI GmbH, Germany, `Torsten.Blochwitz@esi-group.com`
[3]Robert Bosch GmbH, Germany, `Christian.Bertsch@de.bosch.com`
[4]Dassault Systemes GmbH, Germany, `Torsten.Sommer@3ds.com`
[5]Dassault Systemes AB, Sweden, `Karl.Wernersson@3ds.com`
[6]dSPACE GmbH, Germany `{APillekeit,IZacharias,MBlesken}@dspace.com`
[7]PMSF IT Consulting, Germany, `pmai@pmsf.de`
[8]AVL List GmbH, Austria, `Klaus.Schuch@avl.com`
[9]TLK-Thermo GmbH, Germany, `c.schulze@tlk-thermo.com`
[10]Aarhus University, Denmark, `claudio.gomes@ece.au.dk`
[11]Altair, France, `masoud@altair.com`

Abstract

The Functional Mock-up Interface (FMI) (Modelica Association 2021b) is a tool independent standard for the exchange of dynamic models and for co-simulation. FMI 2.0, released in 2014, is recognized as the de-facto standard in industry for exchanging models and tool coupling, and is currently supported by more than 160 simulation tools. Version 3.0 of the standard brings many new features that allow for advanced co-simulation algorithms and new use cases such as packaging and simulation of highly accurate virtual Electronic Control Units (vECUs). Besides Model-Exchange and Co-Simulation, a third interface type, Scheduled Execution, is defined for purely discrete, RTOS-like, simulation and supports preemption. Clocks allow the synchronization of events between Functional Mock-up Units (FMUs) and the importer. There is better support for data types including binary data and arrays. Advanced co-simulation approaches are enabled by intermediate variable access between communication points and allowing event handling. The composition of systems from FMUs is simplified by terminals that can bundle multiple signals. The concept of layered standards allows the extension of the FMI standard.

Keywords: FMI, FMU, Functional Mock-up Interface

1 Motivation

FMI 1.0 (Blochwitz 2011) and FMI 2.0 (Blochwitz 2012) were successfully adopted by industry and are currently supported by more than 160 simulation tools (Modelica Association 2021c). For many years stability was an important success factor of FMI, resulting in maintenance releases of FMI 2.0. However, it became clear that new use cases require improved capabilities that are addressed by the new version of the standard (Modelica Association 2021a), summarized next.

Virtual Electronic Control Units (vECUs). The ability to package control code into Functional Mock-up Units (FMUs) required some workarounds in FMI 2.0. With FMI 3.0, virtual electronic control units (vECUs) can be exported as FMUs in a more natural way using the following new and/or improved features: Terminals (subsection 3.3), clocks (subsection 3.6), new integer types and the new binary type (subsection 3.1), array variables and structural parameters (subsection 3.2), and the new interface type Scheduled Execution (subsection 2.3).

Advanced Co-Simulation. FMI 3.0 introduces, in its Co-Simulation interface type, the Event Mode (subsection 3.4), early return from `fmi3DoStep` (subsection 3.4), and the Intermediate Update Mode (subsection 3.5).

Improved Event Handling across FMUs. The new version of FMI provides an API to enable more flexible event handling and communication: the Synchronous Clocks API (subsection 3.6). For scenarios that are driven by events (e. g., supervisor control systems, engine control systems triggered by crankshaft angle sensors), the clocks API allows FMUs to communicate to the importer detailed information about the timing and cause of events. Moreover, the exact timing that events should happen is communicated unambiguously between FMU and importer, bypassing floating point representation issues. Most of these features are optional to not exclude simulation domains where such features are out of scope and tools that are not able to implement these optional features.

DOI
10.3384/ecp2118117

AI models. In Machine Learning and Artificial Intelligence (AI) new modeling and model training frameworks emerge. More and more, the created models shall interact with established modeling and simulation tools. FMI is a natural means to encapsulate and exchange AI-models with them. This can also lead to hybrid models formed of both physics-based- and AI-models. In order to enable the efficient training of AI-models encapsulated as FMUs, adjoint derivatives are needed, see subsection 3.7.

The following section gives an overview of the different interface types, and the main use cases they apply to. Then, section 3 details the new features. Section 4 introduces some examples that use the new features, and section 5 discusses the measures taken to improve the quality of the standard. Finally, section 6 concludes.

2 Interface Types

FMI 3.0 defines three main interface types: Co-Simulation (CS), Model Exchange (ME), and Scheduled Execution (SE). An FMU may implement one or more of the three interface types. It is a ZIP archive containing: an XML file, describing the model variables and structure; binary and/or source code implementations of the FMI API of the supported interface types; miscellaneous resources; and other related data.

All interface types share common functionality, such as the way variables and clocks are declared/interacted with, or common optional features like store/restore the complete FMU state, or definitions of terminals and icons.

Figure 1 ranks the different interface types according to their simplicity and flexibility trade-offs.

An implementation that interacts with an FMU using the FMI API is called importer.

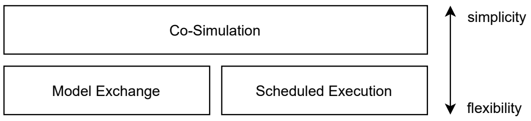


Figure 1. Comparison of interface types

2.1 Model Exchange

The Model Exchange interface exposes a simulation model as a hybrid ordinary differential equation (ODE) to a solver of an importer. Models are described by differential, algebraic and discrete equations, interleaved with time-, state- and step-events. The integration algorithm of the importer is responsible for advancing time, computing state variables, handling events, etc. Figure 2 shows the data flow for Model Exchange.

2.2 Co-Simulation

The Co-Simulation interface is designed both for the coupling of simulation tools, and the coupling of subsystem models, exported by their simulators together with their solvers as runnable code. The data exchange between FMUs is restricted to discrete communication points. In the time between two communication points the subsystem inside an FMU is solved independently by internal means. For FMI for Co-Simulation, the co-simulation algorithm is shielded from how the FMU advances time internally.

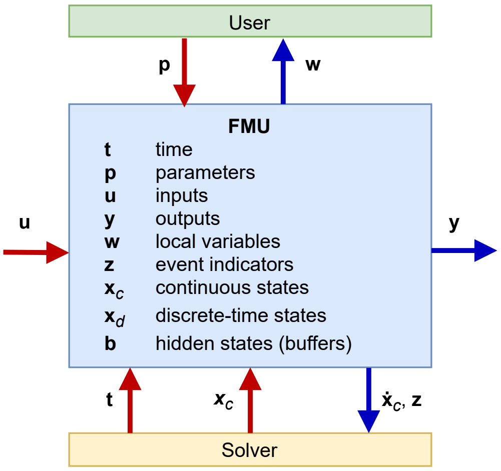


Figure 2. Schematic view of data flow between user, the solver of the importer and the FMU for Model Exchange.

Figure 3 shows the data flow for Co-Simulation.

2.3 Scheduled Execution

The Scheduled Execution interface exposes individual model partitions (e. g., tasks of a control algorithm), to be orchestrated by a scheduler provided by the importer. The scheduler is responsible for advancing the overall simulation time, activating time-based and triggered clocks (for an explanation of clocks see subsection 3.6) for all exposed model partitions of a set of FMUs, and to activate the respective model partition. The Scheduled Execution interface addresses simulation use cases with the following properties, that typically hold when the importer has to communicate with external event sources/sinks that operate on independent individual timing schemes (e. g., real hardware, controller tasks on simulated or real controller units):

1. at any time (even for unpredictable events), an event shall be signaled to an FMU,
2. communication constraints (e. g., execution times, communication deadlines) that are not apparent at FMU simulation level but lead to timing requirements have to be fulfilled by the simulation algorithm,
3. priority information provided by the FMUs has to be evaluated and merged to an overall priority for available model partitions,
4. data shall move between the different FMU model partitions for the same or next activation time.

To address these properties, the Scheduled Execution interface provides support for preemptive multitasking: concurrent computation of model partitions of an FMU

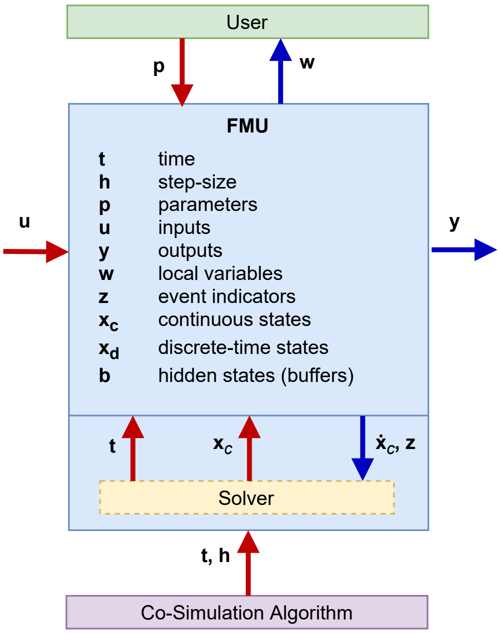


Figure 3. Schematic view of data flow between user, the co-simulation algorithm of the importer and the FMU for Co-Simulation. Compared to Figure 2, the solver is part of the FMU, and not part of the importer.

(i. e., a support of multiple rates) on a single computational resource (e. g., CPU-core). In the rare cases that the FMU has to be able to restrict the preemption for particular code sections, lock/unlock callback functions are provided by the importer. Note that cooperative multitasking for model partitions of an FMU is currently not covered by the interface description, and parallel computation of model partitions is therefore not part of the FMI 3.0 API. However, an FMU may internally use parallel computation on multiple cores, but this results in a binding to a supported operating system.

The FMU must declare the priorities of its model partitions, enabling a global computation order and preemption policy for model partitions across FMUs.

The Scheduled Execution interface has a different timing concept compared to FMI for Co-Simulation: a scheduler's activation of a model partition will compute the results of the model partition defined by an input clock for the current clock tick time t_i (and not for $t_i + h_i$, as defined for `fmi3DoStep` in FMI for Co-Simulation). This is required to handle activations of triggered input clocks which may tick at a time instant that is unpredictable for the simulation algorithm. Typically, hardware I/O or vECU software events belong to this category.

Figure 4 shows the data flow for Scheduled Execution.

3 New Features

3.1 Data Types

FMI 1.0 and FMI 2.0 used a minimal number of numeric data types for interface variables: `fmi2Boolean`, `fmi2Integer` and `fmi2Real`. When packaging numeric codes representing physical processes, these types and the fmi2String type, were sufficient.

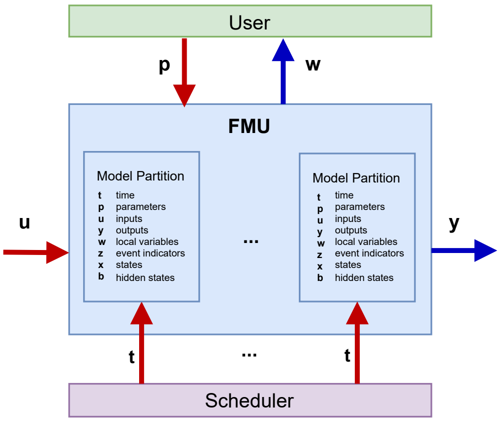


Figure 4. Schematic view of data flow between user, the scheduler of the importer and the FMU for Scheduled Execution

System simulation has enlarged its focus to include controller code (vECUs) to achieve high-quality cyber-physical systems simulations. While packing vECUs as 1.0 or 2.0 FMUs has been practiced since 2010, the restrictions in interface data types introduced noticeable overhead for conversion and copying. Moreover, new automotive applications for system simulation, like Advanced Driver Assistant Systems and Autonomous Drive system components exchange more and more non-numeric data, like object lists, images or even video streams.

Therefore, FMI 3.0 now supports a large set of integer types (signed and unsigned, from 8 to 64 bit) and both 32- and 64-bit floating-point variables. Moreover, binary variables have been introduced to allow the efficient exchange of non-numeric values. Binary variables can be attributed in the `modelDescription.xml` with a `mimeType` to allow proper interpretation of their content.

3.2 Array Variables

In former versions of FMI, only scalar variables were supported; array variables had to be expressed using naming conventions. FMI 3.0 supports array variables natively: Each variable (defined in the `modelDescription.xml`) can have a constant number of dimensions (thus making the variable a multidimensional array variable). The size of a dimension can either be constant or may reference a structural parameter.

A simple example for using arrays would be a generic matrix multiplication FMU. Such an FMU would expose two structural parameters, defining the size of the 1-dimensional input variable, the size of the 1-dimensional output variable and the sizes of the 2-dimensional array (matrix) parameter variable.

A structural parameter can be, like any other parameter, `constant`, `fixed`, or `tunable`. Changes to structural parameters are restricted to special simulation modes:

- Configuration Mode allows changes to `fixed` and tunable structural parameters and can be reached from the mode Instantiated (before Initialization Mode).

- Reconfiguration Mode allows changes to tunable structural parameters. This mode can be reached from Event Mode in Model Exchange, from Step Mode in Co-Simulation and from Clock Activation Mode in Scheduled Execution.

Hence, the sizes of the matrix FMU example do not need to be "constant" after the FMU got generated but could be "fixed" after the FMU gets instantiated or even "tuned" during simulation.

Note that changing the value of a structural parameter might also change the number of continuous states or event indicators.

While the primary use case for structural parameters is for arrays with variable dimension sizes, there can be structural parameters that are not used in dimension elements, e. g., an fmi3String containing a file name to read data from.

FMI 3.0 defines serialization orders for accessing array variables (as well as corresponding derivatives or dependencies).

3.3 Terminals and Icons

Terminals define semantic groups of variables to ease connecting compatible signals on system level. This definition adds an additional layer to the interface description of the FMUs. It does not change the causality of the variables (i. e., inputs and outputs), but enables the definition of physical and bus-like connectors that require special handling on the system level by the importer (e. g., bus frames, flow and stream variables). Terminals can contain terminals to form hierarchies.

The matching rule of a terminal describes the rules for variable matching in a connection of terminals. There are three predefined matching rules: plug, bus, and sequence. The terminal kind can be used to define domain specific member variable sequences, member names and order, or high level restrictions for connections. A member variable always refers to a variable declaration in the ModelVariables element. The variable kind of a terminal member variable defines how the connection of this variable has to be implemented. There are three predefined matching rules: signal (i. e., common signal flow), inflow and outflow (i. e., Kirchhoff's current law). Finally, the concept of stream connectors is utilized (Rüdiger Franke et al. 2009). The TerminalStreamMemberVariable is used for variables which fulfill the balance equation for transported quantities.

To define domain-specific terminals, additional information, like the specific physical meaning of a TerminalMemberVariable or sign conventions, must be standardized. The FMI 3.0 standard itself does not define such domain specific terminals but enables other (layered) standards (see section 3.7) to do so. The XML attributes matchingRule, terminalKind and variableName, are defined as xs:normalizedString and not as xs:enumeration which would restrict their extensibility. The FMI 3.0 specification defines a certain set of values for these attributes (e. g., "signal" or "inflow") and a dedicated semantics. Other standards might introduce other values and other semantics. Importers which do not understand such definitions can ignore them and use the traditional input/output approach specified by the causality attribute of FMI variables. So if a specific vendor definition is not supported, then the importer can ignore the terminal definition and rely on the information in the ModelVariables element.

The graphical representation of the FMU icon can be provided as png file. Additionally a coordinate system, the FMU icon extent, and the placement of each terminal can be defined. The graphical representation of each terminal is also provided as png file. In addition to the png files, svg files can be provided for high quality rendering.

Both, the terminal definitions and the graphical representations are defined in the optional XML file terminalsAndIcons.xml.

3.4 Event handling in Co-Simulation

The Co-Simulation interface of FMI 1.0 and FMI 2.0 is popular because many different simulation mechanisms can be abstracted into one function call fmi2DoStep to let the FMU execute one communication time step. FMI 3.0 extends the Co-Simulation interface with a number of mechanisms to more flexibly control execution of the FMU over time.

Event Mode – The importer can interrupt the Step Mode to transition the FMU into Event Mode. In Event Mode a different set of equations is active inside the FMU and discrete variables may change their values. The importer can solve algebraic loops of the system the FMU is part of and may step the FMU through a series of super-dense time instants, each such step potentially using a different discrete state of the FMU.

Early Return – In order to allow for larger h_i time steps in fmi3DoStep calls, importer and FMU must be able to interrupt such long fmi3DoStep before reaching $t_i + h_i$, in case something "special" happened:

1. The FMU can return from fmi3DoStep early, announcing an internal event and requesting a transition to Event Mode.
2. The importer can request the FMU to return early from fmi3DoStep during Intermediate Update Mode (see next Section) to prevent the FMU from computing beyond an event recently discovered by the importer (e. g., triggered by another FMU).

The FMU indicates via the capability flag hasEventMode if it supports Event Mode.

The importer informs the FMU via argument `eventModeUsed` and `earlyReturnAllowed` of function `fmi3InstantiateCoSimulation` if it supports event handling.

3.5 Intermediate Update Mode

The order of the error in a co-simulation is dominated by the order of the errors made due to the approximation of inputs (Arnold, Clauß, and Schierz 2014). At the same time, higher order input approximation schemes may lead to instabilities, depending on the system being simulated. As such, it is important to provide some degree of control over the input approximations being performed, and allow the importer to obtain some of the intermediate outputs of the FMU. Additionally, the importer may change continuous and discrete input variables between `fmi3DoStep` calls in a way that is hard for the FMU to predict. This causes problems such as excessively small internal solver stepping (due to the discontinuities introduced at communication times) and loss of accuracy (Busch 2016).

FMI 3.0 introduces the Intermediate Update Mode to alleviate these issues, allowing the Importer and FMU to exchange intermediate values for variables. The FMU can call back into the importer to enter this Intermediate Update Mode and ask the importer to update its continuous input variables and allow it to query its continuous output variables (e. g., to supply other FMUs with updates to their inputs). This mechanism replaces `fmi2SetRealInputDerivatives` for input interpolation. This callback mechanism allows the FMU to maintain its internal solver state while new continuous inputs are being set. The FMU can hint to the importer to keep the changes to the continuous input variables within a certain smoothness (see `recommendedIntermediateInputSmoothness`) to optimize convergence.

The Intermediate Update Mode serves a number of other purposes as well:

1. FMUs can inform the importer about pending events.
2. The importer can ask the FMU to return early from an `fmi3DoStep` (see previous Section).
3. In Scheduled Execution, the FMU can inform the importer/scheduler about a clock activation.

The FMU signals via the capability flag `providesIntermediateUpdate` if it supports this feature. The importer provides the pointer to the callback function via argument `intermediateUpdate` of functions `fmi3InstantiateCoSimulation` and `fmi3InstantiateScheduledExecution` as non-NULL if it supports this feature.

3.6 Clocks

System simulation requires the coordination of events across simulation components, in both Model Exchange and Co-Simulation. If these components are packaged as 1.0 or 2.0 FMUs, the importer and FMUs need to use floating point time and epsilon environments with all the known issues to locate such global events. Rüdiger Franke et al. (2017) proposed how to introduce clocks in FMI.

In FMI 3.0, clocks are introduced to allow precise coordination of global events, see Cláudio Gomes et al. (2021). An FMI clock is a special variable that can be active or inactive. When active, the corresponding model partition (a set of equations associated to a clock) becomes active while in Event Mode or Clock Activation Mode.

FMU variables that change only when a specific clock ticks are called clocked variables and are assigned to this clock in the `modelDescription.xml`.

With clocks synchronized across FMUs, algebraic loops can now be solved properly during such global events.

FMI 3.0 distinguishes between time-based clocks and triggered clocks. The latter are raised when something unexpected (e. g., a state event) happens and can be connected to other triggered clocks. The importer forwards that clock activation to the triggered input clocks during Event Mode or Clock Activation Mode.

Time-based clocks come in a few different flavors (see standard document for details) and all require the importer to determine the proper time instant when to activate such clocks - even if the FMU receiving such a time-base input clock defines the period or next event time itself. This is an important distinction to note: the FMU defines the period or next activation, but the importer has the final say at which time instant to actually activate the clock. This is especially important for fixed-step solvers where some flexibility might be required to transition events to one of the communication points.

3.7 Adjoint derivatives

FMI 3.0 offers an additional interface function to calculate partial derivatives. While directional derivatives calculating $v_{\text{sensitivity}} = \mathbf{J} \cdot v_{\text{seed}}$ for the Jacobi matrix $\mathbf{J}$ where already supported in FMI 2.0, now also adjoint derivative calculation is supported by the new interface function `fmi3GetAdjointDerivative`, calculating $v^T_{\text{sensitivity}} = v^T_{\text{seed}} \cdot \mathbf{J}$. They are used, e. g., in AI frameworks, where they are called “vector gradient products” (VGPs). There adjoint derivatives are used in the backpropagation process to perform gradient-based optimization of parameters using reverse mode automatic differentiation (AD). Typically, reverse mode automatic differentiation is more efficient for this use case than forward mode AD, as explained in (Baydin et al. 2015).

3.8 Support for Layered Standards

In order to enable the backward-compatible extension of the FMI standard in minor releases and between minor releases, the FMI project intends the use of the layered standard mechanism to introduce new features in a fully backward-compatible and optional way. A layered standard defines extensions to the base FMI standard by specifying either standardized annotations, standardized extra files in the FMU, and/or support for additional MIME types, e. g., for the interpretation of variables of type `fmi3Binary`.

A layered standard can include a single or combined set of extension mechanisms from this set. The layered standard is thus considered to be layered on top of the definitions and extensions mechanisms provided by the respective FMI base standard.

Layered standards can fall into three categories:

1. Layered standards defined by third parties, without any representations by the FMI project for their suitability or content, or even knowledge by the FMI project about their existence.
2. Layered standards defined by third parties that are endorsed by the FMI project and listed on the FMI project website.
3. Layered standards can be defined/adopted and published by the FMI project itself, making them FMI project layered standards.

Layered standards that have achieved enough adoption or importance to be included into the base standard set could be incorporated into a new minor or major release version of the base standard as an optional or mandatory appendix, making support for this layered standard optional or required for conformance with the newly published minor release version of the base standard.

Examples for layered standards currently developed by the FMI Project will support XCP (ASAM 2021) and Automotive Networks. Further layered standards could define standardized terminals for certain domains.

3.9 Build Configuration for Source Code FMUs

To better support the exchange of FMUs with source code implementations, FMI 3.0 introduces build configurations consisting of an XML document that specifies a set of source files and abstracted build information, like preprocessor definitions, include paths or library dependencies that are needed for building the supplied source code. These can be cross-platform or platform-specific, which gives the importer the ability to choose the correct build configuration for a certain platform.

Here is an example for a build configuration:

```
<BuildConfiguration modelIdentifier="
    PlantModel" description="Build
    configuration for desktop platforms">
<SourceFileSet language="C99">
  <SourceFile name="fmi3Functions.c"/>
  <SourceFile name="solver.c"/>
</SourceFileSet>
<SourceFileSet language="C++11">
  <SourceFile name="model.c"/>
  <SourceFile name="logging/src/logger.c"/>
  <PreprocessorDefinition name="FMI_VERSION"
      value="3"/>
  <PreprocessorDefinition name="LOG_TO_FILE"
      optional="true"/>
  <PreprocessorDefinition name="LOG_LEVEL"
      value="0" optional="true">
    <Option value="0" description="Log infos,
        warnings and errors"/>
    <Option value="1" description="Log
        warnings and errors"/>
    <Option value="2" description="Log only
        errors"/>
  </PreprocessorDefinition>
  <IncludeDirectory name="logging/include"/>
</SourceFileSet>
<Library name="hdf5" version="&gt
    ;=1.8,!=1.8.17,&lt;1.10" external="true"
    description="HDF5"/>
```

4 Examples

In this section, we show some examples where the features, introduced in the previous section, are used. Some of these examples are being developed as reference FMUs, with their source code made available online[1].

4.1 Supervisory Control System

We start with an example that highlights the use of new features of FMI for CS. Consider the example scenario shown in Figure 5, where the FMUs are connected in a typical feedback control loop, with a monitoring and adaptation loop, and the plant has been decoupled into two FMUs. Using the new data types, structured information can be communicated from the controller to the supervisor, and using array variables or terminals, the connections between the controller and plant can be simplified. Thanks to the intermediate value update and early return mechanisms, the rate of sampling of the Controller can be decoupled from the rate of sampling of the supervisor, which in turn can be decoupled from the step size used in the co-simulation.

The intermediate value update enables setting inputs and accessing outputs between the communication points. It can be used to implement advanced co-simulation algorithms to increase stability and reduce the coupling errors between the two plant FMUs. Some examples are given below:

1. The plant inputs can be extrapolated ensuring continuity of signals (Busch 2016).
2. If a conserved quantity such as energy is transported between the plant FMUs, then the additional information enables a reduction of the coupling errors. One available algorithm is the nearly energy-preserving coupling element (Benedikt et al. 2013; Sadjina et al. 2017).
3. Transmission Line Modelling (TLM) systems contain TLM connections which can be interpreted as physically-motivated delayed connections. So introducing a delay between the FMUs, trajectories instead of scalars can be exchanged between the plant FMUs to improve stability and performance (Fritzson, Ståhl, and Nakhimovski 2007; Ochel et al. 2019).

For more advanced co-simulations algorithms, we refer the reader to Cláudio Gomes et al. (2018, Section 4).

[1] https://github.com/modelica/Reference-FMUs

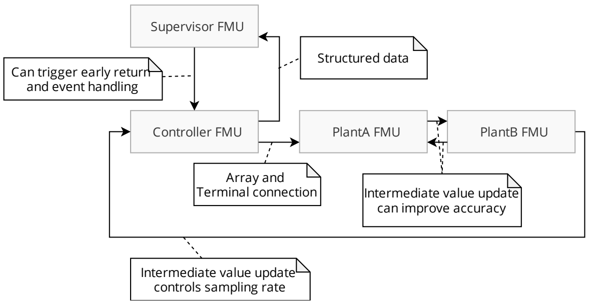

Supervisor FMU
Structured data
Can trigger early return and event handling
Controller FMU
PlantA FMU
PlantB FMU
Array and Terminal connection
Intermediate value update can improve accuracy
Intermediate value update controls sampling rate

Figure 5. Example supervisory control system and how the new features of FMI CS can improve its simulation.

Finally, the Supervisor FMU may reconfigure the Controller FMU when a certain condition is met. When this happens, the Supervisor FMU may signal that stepping phase needs to be halted, and event handing is needed.

4.2 Clocked System

When it is important to make explicit the sampling rate of different subsystems, the Synchronous Clocks API can be used. To illustrate this, consider Figure 6, that shows a variant of the scenario of Figure 5, except the controller FMU has been partitioned across different FMUs. The figure sketches the `CtrlFMU` equations, but note that the importer has no access to these (it can only query the FMU for the values of the output variables). The `CtrlFMU`, every $1/r$ seconds (with r denoting both a clock r and its frequency), gets a sample from the `Plant` (produced by the `Sensor`), and calculates its next state, based on the previous state `pre(u_r)`, the sampled value `x_r`, and some configuration parameter `a` that is calculated by the `Supervisor`. The latter, depending on the `Plant` dynamics, the sampling rate of which we ignore, may decide to reconfigure the `Controller`. By introducing a triggered input clock `s` and a time-based input clock `r`, it is made clear who is responsible for the unambiguous activation of the clocks: the `Supervisor` controls `s`, and the importer controls when to activate `r` (even though, depending on the clock attributes, the `CtrlFMU` may recommend a sample rate that the importer will then obey). Furthermore, no approximate floating point comparisons are needed to know which equations have to be active when entering Event Mode.

We refer the reader to (Cláudio Gomes et al. 2021) and the FMI standard, for more details about Synchronous Clocks.

4.3 Terminals

This section illustrates an exemplary terminal definition of an electrical pin in the FMU XML file.

```
<Terminal name="Pin1" matchingRule="plug">
  <TerminalMemberVariable
   variableKind="inflow"
   memberName="i"
   variableName="Current" />
  <TerminalMemberVariable
   variableKind="signal"
   memberName="v"
   variableName="Voltage" />
</Terminal>
```

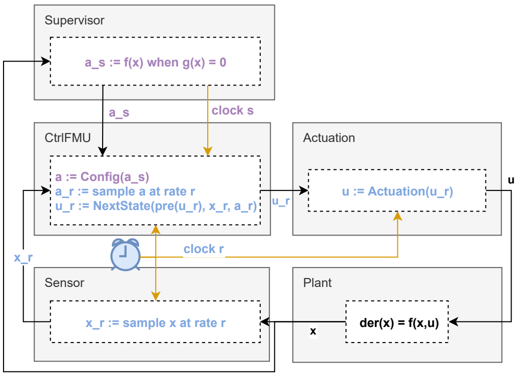


Figure 6. Example clocked control system.

`"Voltage"` and `"Current"` reference to the following FMU-variables:

```
<Float64
  name="Current"
  valueReference="2"
  description="Current output"
  variability="continuous"
  causality="output" />
<Float64
  name="Voltage"
  valueReference="1"
  description="Voltage input"
  variability="continuous"
  causality="input"
  start="0" />
```

The `variableKind` attribute indicates to the importer that Kirchhoff's current law should be applied to the variable `Current`, while `Voltage` should be treated as common signal.

A Modelica tool, for example, can use this XML description to generate a `connector` which automatically matches to `Modelica.Electrical.Analog.Interfaces.Pin` of the Modelica Standard Library since the same names for the member variables are used and `"i"` is marked as flow variable. Similar mechanisms are possible with VHDL-AMS or other tools which support a terminal or bus concept which goes beyond single signals.

Even though the FMU can be imported into acausal tools/languages such as Modelica, the FMU itself is causal. An acausal model can be used to generate several FMUs with different computational causality. The computational causality of the FMU is defined by the causality attributes in the XML file. In the example above `"current"` is always an output of this specific FMU. Cases where the importer prefers a different computational causality than provided by the FMU have to be handled by the importer.

4.4 Virtual Electronic Control Unit

The following example illustrates how FMI 3.0 features can be used for packaging vECUs inside FMUs. It is a reduced version of the example used in the layered standard proposal for automotive network communication.

This layered standard proposal uses clocks, binary variables, naming conventions for variable names and terminals to use FMI 3.0 mechanisms as transport layer for automotive network communication between simulation components. Clocks describe the timing of their respective network frames, to exactly communicate the sending of each frame. Terminals are used to describe the composition of these network frames from PDUs and signals, hierarchically, allowing simplified handling of entire signal groups in system composition tools. Binary signals are used to represent frames in their raw network specific encoding, serializing internal PDUs and signals. For consistent encoding and decoding of these binary signals, standardized network description files are included inside the FMU in the `/extra` directory. Referencing existing standard description files allows reusing existing tools for both exporting and importing such FMUs, while ensuring the same semantics are used for both sender and receiver of signals encoded according to these standards.

The example shows how to describe a CAN message that updates 2 signals, each represented as a `Float32` variable. Naming conventions, described in the layered standard, can be used to match the signals, the corresponding binary variable representing the raw frame data and the clock variable determining the timing of the CAN message (here `POWERTRAIN::tcuSensors_FRAME` and their corresponding triggered `Clock` variable `POWERTRAIN::tcuSensors_CLOCK`). The triggered clock variable controls the time at which a message is set, and should be connected to another clock at the source of the message.

```
<fmiModelDescription fmiVersion="3.0-alpha.6"
 modelName="Network4FMI"
 instantiationToken="Network4FMI">
 <ModelVariables>
  <Float32 name="POWERTRAIN::tcuSensors::
      tcuSensors::vCar"
   valueReference="1001" causality="input"
   variability="discrete" start="0" clocks="
      1004"/>
  <Float32 name="POWERTRAIN::tcuSensors::
      tcuSensors::oilTemp"
   valueReference="1002" causality="input"
   variability="discrete" start="20" clocks=
      "1004"/>
  <Binary name="POWERTRAIN::tcuSensors_FRAME
      "
   valueReference="1003" causality="input"
   variability="discrete" clocks="1004"/>
  <Clock name="POWERTRAIN::tcuSensors_CLOCK"
   valueReference="1004" causality="input"
   variability="clock" interval="triggered"/
      >
...
 </ModelVariables>
</fmiModelDescription>
```

```
<fmiTerminalsAndIcons fmiVersion="3.0-alpha6"
    >
 <Terminals>
  <Terminal terminalKind="bus" name="
      POWERTRAIN" matchingRule="bus"
   description="Powertrain CAN bus defined
      with dbc file">
   <Terminal terminalKind="frame" name="
       tcuSensors" matchingRule="bus">
    <TerminalMemberVariable variableKind="
        signal"
     variableName="POWERTRAIN::
        tcuSensors_FRAME" />
    <TerminalMemberVariable variableKind="
        signal"
     variableName="POWERTRAIN::
        tcuSensors_CLOCK" />
    <Terminal terminalKind="pdu" name="
        tcuSensors" matchingRule="bus">
     <TerminalMemberVariable variableKind="
         signal"
      variableName="POWERTRAIN::tcuSensors
          ::tcuSensors::vCar"
      memberName="vCar" />
     <TerminalMemberVariable variableKind="
         signal"
      variableName="POWERTRAIN::tcuSensors
          ::tcuSensors::oilTemp"
      memberName="oilTemp" />
    </Terminal>
   </Terminal>
  </Terminal>
  ...
 </Terminals>
 <Annotations>
  <Annotation type="ECU" />
 </Annotations>
</fmiTerminalsAndIcons>
```

4.5 Scheduled Execution

In order to illustrate the preemption support, we consider the example of a single FMU, illustrated in Figure 7, where an FMU declares three input clocks and one output clock. Each input clock, when activated, instructs the importer, that acts as a task scheduler, to execute the corresponding model partition as soon as possible.

A model partition, or just partition, represents code that should be scheduled (e.g on a real-time simulator or offline simulator running real-time scenarios) as soon as an input clock ticks. Partitions contain arbitrary code that reads the inputs of the FMU, writes to the FMU's local variables (which can be shared among tasks) and outputs, and can trigger other clocks or update their interval. The inputs to each partition are set by the importer immediately before executing that partition. In Figure 7, u_m^c's partition reads and writes the shared variable x_m, and either updates the interval of v_m^c or ticks y_m^c.

In Figure 7, input clock u_m^c ticks every 10 ms and w_m^c ticks every 50 ms, therefore, every 5th tick, both clocks will tick simultaneously. When that happens, the scheduler needs to know whose task has the highest priority. As a result, the FMU needs to declare a priority level for each input clock. In Figure 7, u_m^c's task (the one executing Partition 1) should be executed before w_m^c's (Partition 3).

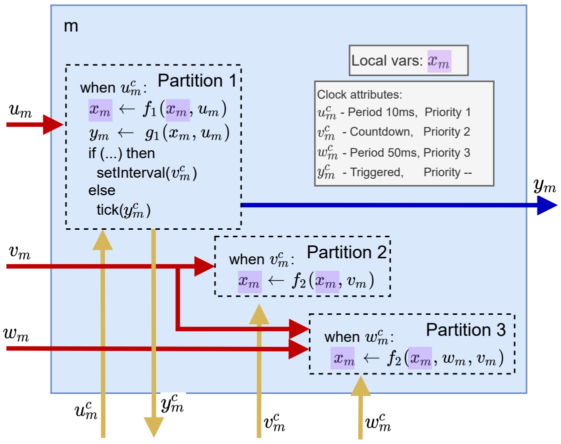

m
Local vars: x_m
when u_m^c: Partition 1
$x_m \leftarrow f_1(x_m, u_m)$
$y_m \leftarrow g_1(x_m, u_m)$
if (...) then
setInterval(v_m^c)
else
tick(y_m^c)
Clock attributes:
u_m^c - Period 10ms, Priority 1
v_m^c - Countdown, Priority 2
w_m^c - Period 50ms, Priority 3
y_m^c - Triggered, Priority --
u_m
y_m
v_m
when v_m^c: Partition 2
$x_m \leftarrow f_2(x_m, v_m)$
w_m
when w_m^c: Partition 3
$x_m \leftarrow f_2(x_m, w_m, v_m)$
u_m^c
y_m^c
v_m^c
w_m^c

Figure 7. Example FMU implementing the SE interface.

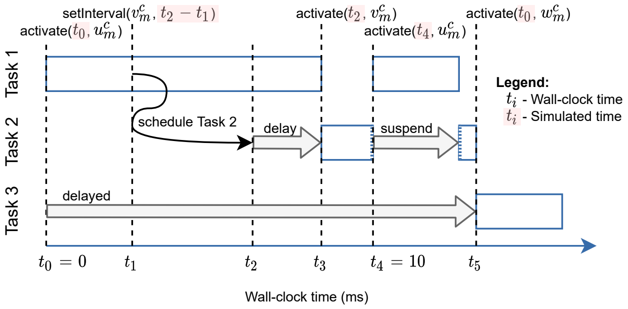


Figure 8. Example execution trace of Figure 7. Task *N* corresponds to execution of partition *N*, as detailed in Figure 7.

Because tasks can be preempted, in rare cases, the FMU has to be able to restrict the preemtion for particular sections, such as updating a shared variable. As such, the FMU must inform the importer of when it should not be interrupted, to avoid race conditions. Since partitions can trigger and update the interval of other clocks, the FMU may use the Intermediate Update Mode, in the middle of the calculation of a partition, to inform the importer that a clock is about to tick or has a new interval, so that the importer can schedule the corresponding tasks.

Figure 8 illustrates a possible execution trace of the tasks corresponding to the partitions declared in Figure 7. At the initial wall-clock time, both Task 1 and 3 are scheduled to execute. Since Task 1 has higher priority, it runs first, and Task 3 is delayed. While executing Task 1, the FMU informs the importer that v_m^c's task (Task 2) should be scheduled to run at wall-clock time t_2. At wall-clock time t_2, Task 1 is still executing, so Task 2 is delayed until wall clock time t_3. At wall-clock time t_3, Task 2 starts executing, but note that the activation time of Task 2 is still its scheduled time t_2. This is where the wall-clock time t_3 differs from the simulated time t_2. At t_4, Task 2 is preempted, because of Task 1. Finally, after being delayed substantially, Task 3 gets to execute, with its simulated time t_0.

We refer the reader to (Cláudio Gomes et al. 2021) and the FMI standard, for more details about the Scheduled Execution API.

5 Quality Improvement Measures and Prototypical Implementations

The development of new features followed the FMI development process (Modelica Association 2015) with the creation of FMI Change Proposals (FCP) providing the use cases, suggested changes and partial prototypes.

During the development of the FMI 3.0 standard, the text was completely restructured and several concepts were unified between the different interface types in a common concepts section. The state machines were unified between the different interface types.

In order to streamline the standard text, implementation specific hints will be singled out in an *FMI implementers guide*.

Reference FMUs (*Reference FMUs* 2021) were created to showcase and test certain features of FMI. These FMUs and code snippets extracted from them, are continuously compiled and example XML files automatically validated against the schema files in a continuous integration environment to ensure correctness of the examples included into the standard document.

Several tools were used to validate prototypes of FMI 3.0 features: Altair Activate, Dymola, fmpy (*fmpy* 2021), Model.CONNECT™, Silver, SimulationX, among others.

6 Summary and Outlook

The success of the Modelica Association's FMI standard 1.0 and 2.0 has created the desire to improve simulation efficiency and accuracy, as well as to enable new use cases. FMI 3.0 introduces a number of new features and improvements to address many of the needs found in current industrial and research applications. More importantly, opening up to layered standards will help to address many of the future needs not yet envisioned and to address industry specific requirements best addressed by special-purpose extensions without weighting down the core FMI standard document.

The developers of the current version of FMI are well aware of the current challenges of the simulation community. In particular, simulation efficiency will be the focus of future FMI development.

Another important drive for future FMI versions is harmonization with other complementary Modelica Association standards, such as SSP, DCP and eFMI. Finding structural ways to make these standards more compatible and therefore easier to implement, will increase each standard's added value.

Acknowledgements

The authors wish to thank all the contributors to the FMI specification. Contributions of ESI ITI GmbH were carried out within the project FMI4BIM, funded by the German Federal Ministry for Economic Affairs and Energy.

Supported by:

Federal Ministry
for Economic Affairs
and Energy

on the basis of a decision
by the German Bundestag

References

Arnold, Martin, Christoph Clauß, and Tom Schierz (2014). "Error Analysis and Error Estimates for Co-Simulation in FMI for Model Exchange and Co-Simulation v2.0". In: *Progress in Differential-Algebraic Equations*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 107–125. ISBN: 978-3-662-44926-4. DOI: 10.1007/978-3-662-44926-4_6.

ASAM (2021). *ASAM MCD-1 XCP Standard*. URL: https://www.asam.net/standards/detail/mcd-1-xcp/wiki/ (visited on 2021-04-18).

Baydin, Atilim Gunes et al. (2015). *Automatic differentiation in machine learning*. URL: https://arxiv.org/abs/1502.05767 (visited on 2021-05-06).

Benedikt, M et al. (2013-06). "NEPCE-A Nearly Energy Preserving Coupling Element for Weak-Coupled Problems and Co-Simulation". In: *IV International Conference on Computational Methods for Coupled Problems in Science and Engineering, Coupled Problems*. Ibiza, Spain, pp. 1–12.

Blochwitz, Torsten et al. (2011). "The Functional Mockup Interface for Tool independent Exchange of Simulation". In: *8th International Modelica Conference*. URL: http://www.ep.liu.se/ecp/063/013/ecp11063013.pdf.

Blochwitz, Torsten et al. (2012). "Functional Mockup Interface 2.0: The Standard for Tool Independent Exchange of Simulation Models". In: *8th International Modelica Conference*. URL: https://lup.lub.lu.se/search/ws/files/5428900/2972293.pdf.

Busch, Martin (2016-09). "Continuous Approximation Techniques for Co-Simulation Methods: Analysis of Numerical Stability and Local Error". In: *Journal of Applied Mathematics and Mechanics* 96.9, pp. 1061–1081. ISSN: 00442267. DOI: 10.1002/zamm.201500196.

fmpy (2021). URL: https://github.com/CATIA-Systems/FMPy (visited on 2021-04-20).

Franke, Rüdiger et al. (2009). "Stream Connectors – An Extension of Modelica for Device-Oriented Modeling of Convective Transport Phenomena". In: *Proceedings of the 7th International Modelica Conference* (Como, I, September 20–22, 2009). Ed. by Francesco Casella. Linköping Electronic Conference Proceedings. Linköping: Linköping University Electronic Press, pp. 108–121. DOI: 10.3384/ecp09430078. URL: http://dx.doi.org/10.3384/ecp09430078.

Franke, Rüdiger et al. (2017). "Discrete-time models for control applications with FMI". In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. 132. Linköping University Electronic Press, pp. 507–515. URL: https://2017.international.conference.modelica.org/proceedings/html/submissions/ecp17132507_FrankeMattssonOtterWernerssonOlssonOchelBlochwitz.pdf.

Fritzson, Dag, Jonas Ståhl, and Iakov Nakhimovski (2007). "Transmission Line Co-Simulation of Rolling Bearing Applications". In: *48th Conference on Simulation and Modelling*. Göteborg, Sweden: Citeseer, pp. 24–39.

Gomes, Cláudio et al. (2018). "Co-Simulation: A Survey". In: *ACM Computing Surveys* 51.3, 49:1–49:33. DOI: 10.1145/3179993.

Gomes, Cláudio et al. (2021). "The FMI 3.0 Standard Interface for Clocked and Scheduled Simulations". In: *Proceedings of the 14th International Modelica Conference*. 14th International Modelica Conference. online: Linköping University Electronic Press, Linköpings Universitet, to be published.

Modelica Association (2015-07). *FMI Development Process And Communication Policy*. URL: https://github.com/modelica/fmi-standard.org/blob/master/assets/FMI_DevelopmentProcess_1.0.pdf.

Modelica Association (2021a-04). *Functional Mock-up Interface Specification, v3.0beta.1*. URL: https://github.com/modelica/fmi-standard/releases/tag/v3.0-beta.1.

Modelica Association (2021b). *FMI Website*. URL: https://fmi-standard.org/ (visited on 2021-04-18).

Modelica Association (2021c). *FMI Website*. URL: https://fmi-standard.org/tools/ (visited on 2021-04-18).

Ochel, Lennart et al. (2019-02). "OMSimulator - Integrated FMI and TLM-Based Co-Simulation with Composite Model Editing and SSP". In: *The 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*, pp. 69–78. DOI: 10.3384/ecp1915769.

Reference FMUs (2021). URL: https://github.com/modelica/Reference-FMUs (visited on 2021-05-02).

Sadjina, Severin et al. (2017-07). "Energy Conservation and Power Bonds in Co-Simulations: Non-Iterative Adaptive Step Size Control and Error Estimation". In: *Engineering with Computers* 33.3, pp. 607–620. ISSN: 1435-5663. DOI: 10.1007/s00366-016-0492-8.

The FMI 3.0 Standard Interface for Clocked and Scheduled Simulations

Cláudio Gomes[1] Masoud Najafi[2] Torsten Sommer[3] Matthias Blesken[4] Irina Zacharias[4]
Oliver Kotte[5] Pierre R. Mai[6] Klaus Schuch[7] Karl Wernersson[8] Christian Bertsch[5]
Torsten Blochwitz[9] Andreas Junghanns[10]

[1]Department of Electrical and Computer Engineering, Aarhus University, Denmark, claudio.gomes@ece.au.dk
[2]Altair, France, masoud@altair.com
[3]Dassault Systemes GmbH, Germany, Torsten.SOMMER@3ds.com
[4]dSPACE GmbH, Germany, {MBlesken,izacharias}@dspace.de
[5]Corporate Research, Robert Bosch GmbH, Renningen, Germany,
{Oliver.Kotte,Christian.Bertsch}@de.bosch.com
[6]PMSFIT, Germany, pmai@pmsfit.de
[7]AVL, Austria, klaus.schuch@avl.com
[8]Dassault Systemes AB, Sweden, karl.wernersson@3ds.com
[9]ESI ITI, Germany, Torsten.Blochwitz@esi-group.com
[10]Synopsys, Germany, Andreas.Junghanns@synopsys.com

Abstract

This paper gives an overview of the FMI 3.0 support for two kinds of clock-based simulations: Synchronous Clocked Simulation, and Scheduled Execution. The former is used when the information about multiple simultaneous events (cause and exact time of occurrence) can be unambiguously conveyed. The later facilitates real-time simulations comprising multiple black-box models, by allowing fine grained control over the computation time of sub-models. A formalization is presented along with example application scenarios, meant as an introduction to the conceptualization of clocks in the FMI Standard.

Keywords: functional mockup interface, synchronous clocks, reactive systems, real-time simulation, scheduling, real-time operating system.

1 Introduction

As more and more Modeling and Simulation (M&S) tools are used in system engineering processes, it becomes clear that standards are needed to improve the interoperability of such tools. The Functional Mockup Interface (FMI) Standard (2.0 2014) aims at enabling the exchange and cooperative simulation of black-box models. Version 2.0 of the standard strikes a balance between supporting the most common features across the plethora of M&S tools, and enabling the advanced simulation scenarios. Its wide adoption has, however, placed pressure in supporting two important use cases are: simulation scenarios where timed and state events play a frequent role in synchronizing a subset of the participating models (e.g., controller code with tasks running at different rates); and scenarios where the goal is to control the computation time of the different models, so that a real-time co-simulation can be achieved.

Contribution. This paper gives an overview of the FMI 3.0 support for two kinds of clock based simulations: Synchronous Clocked Simulation (SC), and Scheduled Execution (SE). The former aims at scenarios where the cause and exact time of occurrence of multiple simultaneous events can be unambiguously conveyed. The later facilitates real-time simulation among black-box models, by allowing a finer grained control (compared to version 2.0) over when/which model partitions can be executed.

Structure. The next section will introduce the common concepts and the interface elements that are common to SC and SE. Then in Section 3 SC is detailed, along with a motivating example. Section 4 focuses on SE, following the same structure as Section 3. In Section 5, we discuss some of the relevant related works, and in Section 6 we summarize and conclude.

2 Common Interface and Concepts

Co-simulation is a technique to combine multiple black-box simulation units to compute the combined models' behaviour. See Kübler and Schiehlen (2000) and Gomes et al. (2018), for an introduction. The simulation units, often developed and exported independently from each other in different M&S tools, are coupled using an orchestration algorithm, often developed independently as well, that communicates with each simulation unit via its interface. This interface, an example of which is the FMI Standard interface for Co-Simulation, comprises functions for setting/getting inputs/outputs and computing the associated model behaviour over a given time interval.

The FMI 3.0 defines three interface types: the Co-Simulation (CS), the Model Exchange (ME), and the Scheduled Execution (SE). In the FMI, a simulation unit is called a Functional Mockup Unit (FMU), and it may implement one or more of the three interfaces. An FMU is a zip containing: binaries and/or source code implementing the API functions; miscellaneous resources; and an XML file, describing the variables, model structure, and other data.

For each interface type, the FMU may implement optional features, such as declaring synchronous clocks (in case of ME or CS), or scheduled execution clocks (in case of SE). Figure 1 summarizes the different interface types and the main concepts relevant to this paper. All three interfaces (CS, ME, and SE) share common functionality, such as the declaration and usage of variables and clocks.

The differences between the three interface types can be seen on the left hand side of Figure 1. The Importer refers to the software that imports the FMU. We distinguish three importers, each corresponding to one of the interface types, and each with different responsibilities. The ME importer often needs to provide a differential equation (ODE) solver, and must handle events. In contrast, the CS importer does not need to provide an ODE solver, because such a solver can be implemented inside the CS FMU. Finally, the SE importer needs a task scheduler that will determine exactly when each task implemented in the FMU will be executed.

The ME and CS both contain mechanisms to communicate events to the importer, and, as we detail later, both enable Synchronous Clocked (SC) simulation.

Broadly, a simulation involving multiple connected FMUs goes through the following modes[1]:

Initialize – The FMUs are instantiated and their initial state/inputs/outputs/parameters are calculated or set by the importer.

Step – The simulation is progressing in simulated time, and FMUs that represent ODEs are being numerically integrated.

Event – The simulated time is stopped and events (e.g. clock ticks, parameter changes) are being processed.

Terminate – The simulation has finished and all resources are freed.

The Step and Event modes come after the Initialize mode, and are interleaved.

In the following sub-sections, we introduce FMI3.0 clocks, how they are declared, connected, and interacted with, as well as common constraints imposed by the standard. These are common to the SC and SE clock interpretations.

2.1 Clock Taxonomy

Clocks represent an abstraction of activities whose occurrence is tied to specific points in time. They appear in many modeling formalisms for systems that interact with the real world (Benveniste et al. 2003; Modelica Association 2021), where it is important to represent computations that happen at different rates, or as a result of conditions observed in the environment. Conceptually, each clock represents a sequence of instants in time where the clock is active, called ticks. From the entities that can interact with a clock, we highlight the FMU and the importer (recall Figure 1). The FMU is the entity that declares the clock, while the importer is the code activating the clock in the FMU.

Clocks are declared in the XML file, and can be seen as a special kind of variable. The XML description for each clock contains, among others, an identifier called the value reference, a causality attribute (whether the clock is an input or output, as we will discuss later), and an interval attribute (declaring the type of clock, discussed later). Dynamically, during the simulation, each clock can be either active or inactive (denoted as the clock's state), and its state can be either set or get by the importer, depending on the clock type and its causality (see below).

There are two main types of clocks: time-based and triggered. Time-based clocks are associated with an interval, dictating, at any moment in simulated time, the interval (in simulated time units) between the last tick and the next tick. Such intervals can be queried or set by the importer, depending on the clock's interval attribute (see below). In contrast, triggered clocks have no a priori known interval. The FMU or importer has to set/get the (activation) state of the clock. The different clock types are listed in Table 1 according to who calculates the intervals and ticks the clock.

Before discussing the causality of clocks, it is important to distinguish between the entity that dictates the clock interval vs. the entity that actually activates the clock. This distinction is important in the context of the FMI because the simulated time is a real-valued quantity, represented by a finite-resolution variable. For example, the FMU may declare the interval of a periodic clock in the XML, but it is the importer that will decide exactly at which simulated time the clock ticks. Due to numerical inaccuracies, it may happen that the interval (in simulated time) between clock ticks does not match exactly the interval declared by the FMU.

Time-based clocks are always input clocks, since it is always the importer that is responsible for activating the clock (even though the clock interval information may come from other entities, as shown in Table 1). Triggered clocks, on the other hand, can be input or output clocks. Triggered input clocks, just like time-based input clocks, can only be set by the importer, whereas triggered output clocks are set internally by the FMU, and can only be queried by the importer. The causality therefore plays a role in determining how clocks can be connected.

2.2 Clock Variables and Dependencies

Just like any other variable, an output clock can be connected to an input clock. It is also possible to connect two input clocks or even have one input clock connected to two different output clocks. A connection from clock

[1] This is a simplification of the states or modes defined in the state diagrams of the FMI 3.0 standard.

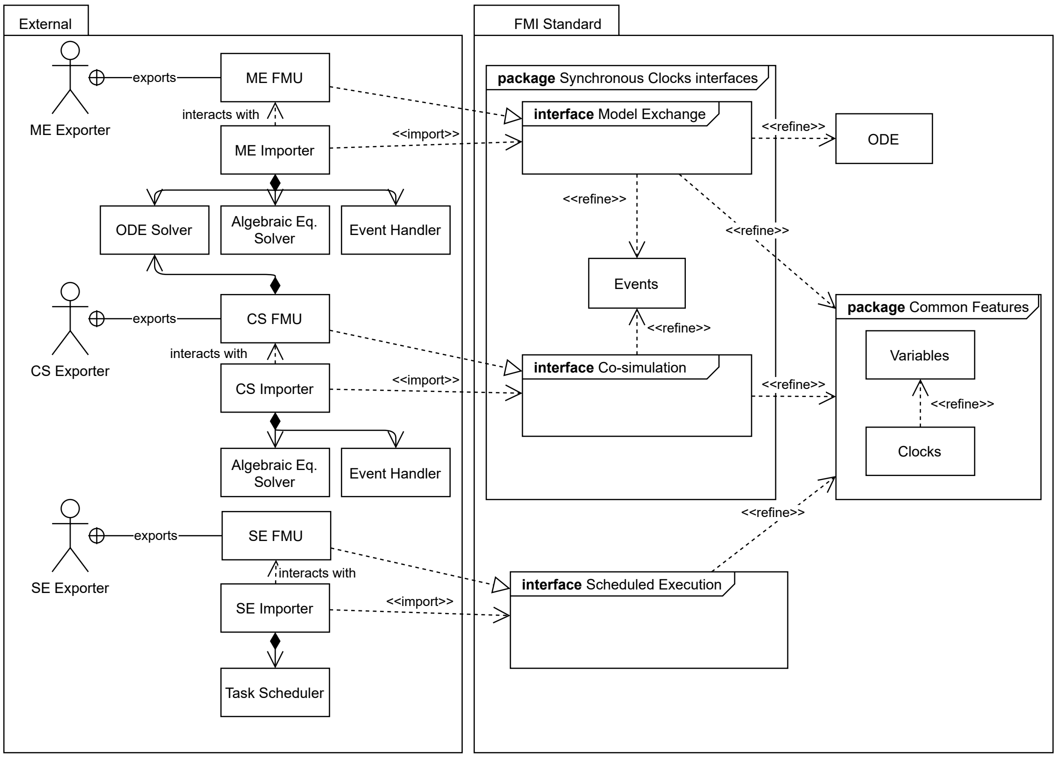


Figure 1. Overview of relevant concepts. Note that there might be domain specific importers which do not need an ODE solver because the supported FMUs do not contain continuous variables. This figure attempts to illustrate the most common differences between the interface types.

w^c to clock v^c means that whenever clock w^c ticks, then clock v^c should also tick. For triggered clocks, that is relatively easy to enforce: whenever one clock activates, the other should be activated. For time-based clocks, the importer must take into account the interval attributes of the clocks and decide whether such connection makes sense or not. For example, if one clock has a constant interval, and another clock has a fixed interval, then the importer may simply set the correct period for the second clock.

FMUs can declare the internal dependencies between their output and their input variables in the XML section denoted as Model Structure. An output variable y depends on an input variable u when the computation of y's value requires the value of u. For example, in Figure 2, y_m is computed from, among other dependencies, u_m.

Each output clock y^c can also depend on one or more input clocks or variables. The meaning is that the state of such input clocks or the value of the input variables is taken into account when deciding whether y^c will tick. For example, in Figure 2, y^c_m *may* tick when u^c_m ticks, or because of the value change of u_m. Note that it is not necessarily the case that y^c will tick whenever an input clock, that y^c depends on, ticks.

When a clock w^c ticks (we use w^c when the causality of the clock is irrelevant), there is a set of variables whose values are computed. We denote that set by "w^c's variables", or "clocked variables" when the specific clock is unimportant. FMI imposes few constraints on the clocked variables. However, the FMU can declare in its XML, for each variable, which clocks w^c depends on (usually one). For example, in Figure 2, y_m is computed when u^c_m ticks. Lacking such declaration, the importer needs to assume the worst case: all output variables are computed when w^c ticks. The value of w^c's variables should only be accessed when (one of the) w^c is active, i.e., is ticking. Accessing the w^c's variables when w^c is not ticking results in undefined behaviour.

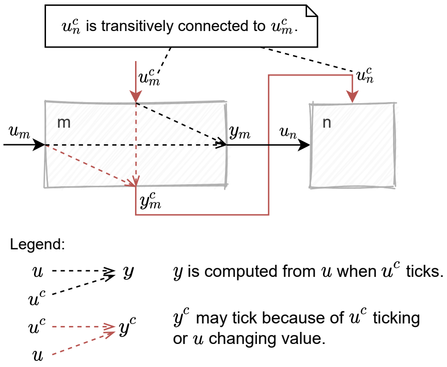

u^c_n is transitively connected to u^c_m.
u^c_m
u^c_n
m
n
u_m
y_m
u_n
y^c_m
Legend:
u
y
u^c
y is computed from u when u^c ticks.
u^c
y^c
u
y^c may tick because of u^c ticking or u changing value.

Figure 2. Example clock connections and dependencies. The symbols m and n refer to FMUs.

3 Synchronous Clocked Simulation

In this section, we describe the Synchronous Clock (SC) interpretation of the clocks interface, introduced in the previous section. This interpretation is inspired by the

Table 1. Overview of clock types and their attributes.

Clock Type	Period	Interval	Interpretation
time-based	periodic	constant	FMU declares period in XML.
		fixed	Importer sets the interval during Initialize.
		calculated	FMU calculates period in Initialize mode.
		tunable	FMU calculates period in Event mode (CS) or after executing model partition (SE).
	aperiodic	changing	FMU calculates interval after each clock tick.
		countdown	FMU calculates interval after an event.
triggered	–	triggered	There's no known interval. The clock ticks unpredictably, either due to FMU current state/inputs, or due to events.

clock implementation in the Modelica specification (Modelica Association 2021) and existing synchronous clock theories such as Benveniste et al. (2003), but had to be adapted to reflect the constraints of black-box co-simulation. As such, we offer no guarantees of semantic equivalence.

We start with detailing the main simulation modes for both ME and CS FMUs, as if no clocks were declared. In order to focus on the essential mechanisms, we abstract away from the ME and CS interfaces, and present them in a unified manner using set theoretic constructs, while referring the reader to the standard for more details.

3.1 Background on CS and ME

Following the super-dense time formulation as in Lee and Zheng (2005), the simulation time is a tuple $t = (t_R, t_I)$ where $t_R \in \mathbb{R}_{\geq 0}$, $t_I \in \mathbb{N}_{\geq 0}$. In Step mode, the real part of time t_R is increasing and $t_I = 0$, and during Event mode, the integer part of time t_I is increasing while t_R is held constant. Figure 3 illustrates a possible trajectory for the values of a variable v under super-dense time. As can be seen, the Step mode produces a continuous evolution for the value of v, while the Event mode introduces discontinuities in the calculation of v. Under Event mode, a variable may acquire multiple values, each computed by one iteration of the Event mode, discerned by the t_I part of the timestamp.

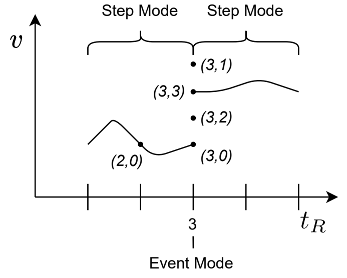


Figure 3. Example variable trajectory under super-dense time.

In Step mode, the FMU and importer cooperate in approximating the solution of a system of differential equations, described by the FMU. In the case of ME, the FMU provides the derivatives and the importer provides the inputs and solver, whereas in CS, the importer provides the inputs, and the FMU provides the derivatives and solver (recall Figure 1).

The importer may then switch the FMU to Event mode if one or more of the following situations occur[2]:

Time events – the simulated time $t = (t_R, 0)$ reached a value t_R that was known at the end of the last Event mode;

State events – The value of some variable crossed a threshold that is known to the FMU;

Input events – The value of an input variable changed in a discrete way, introducing a discontinuity.

In version 3 of the FMI Standard, both ME and CS interfaces describe the mechanism by which the FMU communicates the occurrence of events to the importer, so we will not discuss these mechanisms here. It suffices to assert that the importer is able to determine that the FMU should switch to Event mode at the appropriate simulated time.

During Event mode, the FMU and importer cooperate in solving a set of algebraic equations that are associated with the event that triggered the Event mode, known to the FMU. To solve the equations, the importer will typically construct a dependency graph between the output and input variables, using the Model Structure declared by the FMU. Note that the FMU may be part of a larger simulation model, where external variables form its inputs, and can also depend on its outputs. Therefore the dependency graph may involve not just the FMU variables, but other relevant external variables. As a result, there might exist cyclic dependencies between variables of the FMU. These manifest in the form of non-trivial strongly connected components in the dependency graph (Tarjan 1971). It is up to the importer to solve the algebraic loop, by setting and querying the variables of the FMU. The FMU plays its role by recomputing any output variable that might change

[2] There are other kinds of events, but for simplicity we highlight the main ones.

as a result of new values for the input variables set by the importer. The important outcome is that all variables of the FMU have acquired a value.

The FMU may remain in Event mode, and perform a new iteration, if more events occur. These new events may be caused by the importer or by a new value for some variable. The FMI defines the mechanism by which the FMU or importer agree that a new event iteration is needed. Each new event iteration corresponds to one increment in the integer part of the simulated time. If no more event iterations are needed, Event mode is finished.

Note that, in Event mode, as part of the procedure to solve non-linear equations, there may be hundreds of iterations to converge and obtain a solution. These intermediate values are not shown in Figure 3 and do not cause the integer part of super-dense time to increment because they happen within one super-dense time instant. Therefore in Figure 3 there are three event handling iterations. When switching back to Step mode, the FMU also informs the importer of the next time-based event (if such event is defined).

3.2 Discerning Events

The basic event signaling mechanism offered by FMI 2.0 is adequate for most applications that do not rely on many events. However, they are insufficiently expressive for simulations with many simultaneous events. We illustrate this with a simple example shown in Figure 4, devised to motivate the need for clocks. The example shows a closed loop control system, where the `CtrlFMU` is specified as an FMU, and the remaining sub-models are specified in some other language. We sketch the `CtrlFMU` equations, but note that the importer has no access to these (it can only query the FMU for the values of the output variables). The `CtrlFMU`, every $1/r$ seconds (we abuse the notation r to denote both a clock r and its frequency), gets a sample from the `Plant` (produced by the `Sensor`), and calculates its next state, based on the previous state `pre(u_r)`, the sampled value `x_r`, and some configuration parameter `a` that is calculated by the `Supervisor`. The latter, depending on the `Plant` dynamics, the sampling rate of which we ignore, may decide to reconfigure the `Controller`.

Using only the basic event mechanism of FMI, it is cumbersome to simulate Figure 4, for the following reasons:

- If it is the FMU that decides when to sample, there is no way for the importer to know the sample rate r. The importer only receives information about the next time event, after each Event mode of the `CtrlFMU`.
- There is no way for the FMU to know exactly which equations to enable when entering Event mode. When the `Supervisor` computes a new value for `a_s`, the `CtrlFMU` must be in Event mode, because of the input event. Then `CtrlFMU` must rely on approximate floating point comparisons to know that only the `Config` equation is to be enabled. Conversely, when a new sample `x_r` is available, the `CtrlFMU` must know that the `Config` equation must remain disabled.

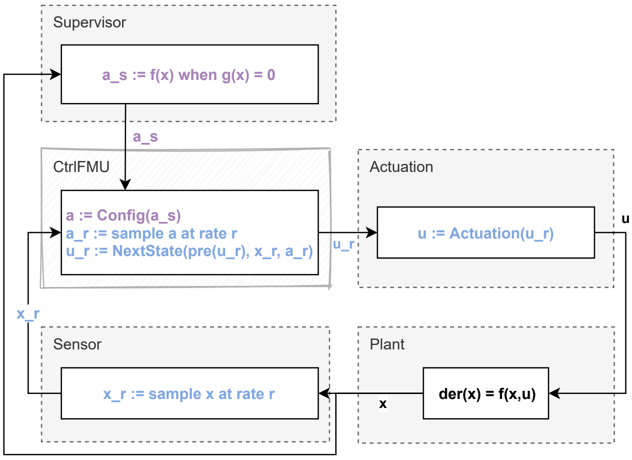

Supervisor
a_s := f(x) when g(x) = 0
a_s
CtrlFMU
a := Config(a_s)
a_r := sample a at rate r
u_r := NextState(pre(u_r), x_r, a_r)
u_r
Actuation
u := Actuation(u_r)
u
x_r
Sensor
x_r := sample x at rate r
x
Plant
der(x) = f(x,u)

Figure 4. Motivating example with supervisor controller.

Figure 5 shows how clocks address the limitations highlighted by the example in Figure 4. By introducing a triggered input clock `s` and a time-based input clock `r`, it is made clear who is responsible for the unambiguous activation of the clocks: the `Supervisor` controls `s`, and the importer controls `r`. Furthermore, no approximate floating point comparisons are needed to know which equations have to be active when entering Event mode.

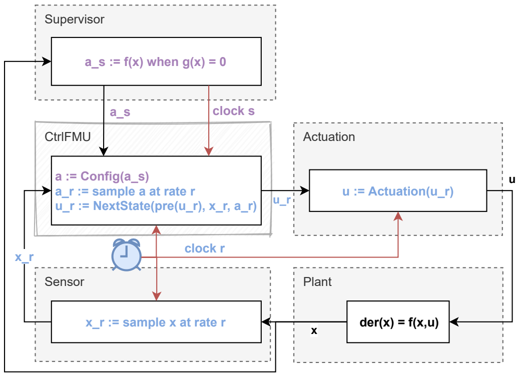


Figure 5. Clocked version of Figure 4.

3.3 Synchronous Clocks Semantics

We now detail the main functions that interact with clocks. In order to do so, we must define a compact FMU abstraction. Without loss of generality, we can focus on formalizing what happens in a simulation where the FMU is in Step mode, switches to Event mode, and then resumes Step mode.

Definition 1 (SC FMU Instance)**.** An SC FMU instance with identifier m is represented by the tuple

$$\langle S_m, U_m, Y_m, U_m^c, Y_m^c, \texttt{set}_m, \texttt{get}_m, \\ \texttt{set}_m^c, \texttt{get}_m^c, \texttt{commit}_m^c, \texttt{stepT}_m, \texttt{stepE}_m, \texttt{nextT}_m\rangle$$

where:

- S_m represents the abstract set of possible FMU states. A given state $s_m \in S_m$ of m represents the complete internal state of m: active clocks, active equations, current mode (Step or Event mode) current valuations for input and output variables, etc.
- U_m and Y_m represent the set of input and output variables, respectively.
- U_m^c and Y_m^c represent the set of input and output clocks, respectively.
- $\texttt{set}_m : S_m \times U_m \times \mathscr{V} \to S_m$ and $\texttt{get}_m : S_m \times Y_m \to S_m \times \mathscr{V}$ are functions to set the inputs and get the outputs, respectively (we abstract the set of values that each input/output variable can take as $\mathscr{V}$). Both $\texttt{set}_m$ and $\texttt{get}_m$ return a new state because both can trigger the computation of equations.
- $\texttt{set}_m^c : S_m \times U_m^c \times \mathbb{B} \to S_m$ and $\texttt{get}_m^c : S_m \times Y_m^c \to S_m \times \mathbb{B}$ are the functions that (de-)activate the input clocks and query the output clocks (returning the activation status), respectively, and $\mathbb{B}$ denotes the boolean set.
- $\texttt{commit}_m^c : S_m \times W_m^c \to S_m$ is a function that updates the clocked states of a given input/output clock in the set $W_m^c = U_m^c \cup Y_m^c$. Clocked states are clocked variables whose value depends on the previous value (e.g., u_r in Figure 4).
- $\texttt{stepT}_m : S_m \times \mathbb{R}_{\geq 0} \to S_m \times \mathbb{R}_{\geq 0} \times \mathbb{B}$ is a function representing the Step mode computation. If m is in state s_m at simulated time (t_R, t_I), $(s_m', h, b) = \texttt{stepT}_m(s_m, H)$ approximates the state s_m' of m at time $(t_R + h, 0)$, with $h \leq H$. When $b = true$, we know that the importer and m have agreed to interrupt the Step mode prematurely, and m is ready to go into Event mode.
- $\texttt{stepE}_m : S_m \to S_m \times \mathbb{B}$ represents one super-dense time iteration of the Event mode. If m is in state s_m at time (t_R, t_I), then $(s_m', b) = \texttt{stepE}_m(s_m)$ represents the computation of m's internal super-dense step transition, where s_m' represents the state at $(t_R, t_I + 1)$ and b informs the importer whether one more Event iteration is needed.
- $\texttt{nextT}_m : S_m \times U_m^c \to \mathbb{R}_{\geq 0} \cup \{NaN\}$ is the function that allows the importer to query the time of the next clock tick. This function is only applicable to tunable, changing, and countdown clocks, and the returned value is calculated according to the clock type as discussed in Table 1. The value NaN can be returned for countdown clocks, and it means that the clock currently has no schedule.

The major differences between above formalization and the FMI interface are as follows.

- There is no explicit representation of state. Most FMI functions take an FMU instance as an argument, and the manipulations to the instance are performed implicitly. We choose to make state explicit so as to explicitly convey which functions change the state of the instance.
- The FMI describes the callback function by which, in CS, the FMU and importer may decide when to prematurely terminate the invocation to $\texttt{stepT}_m$. For ME, the importer is responsible for implementing $\texttt{stepT}_m$ (recall Figure 1).

We now discuss informally the semantics of each function implemented in an FMU instance m, with a focus on the clock functions. Since clock operations happen only in Event mode, we will focus on that mode. Moreover, we present the semantics in the order that a generic importer would interact with the FMI instance m.

Entering Event Mode. Clocks can only tick in Event mode. During Step mode, the FMI provides mechanisms for the FMU and importer to agree that there's a clock that needs to tick, and will therefore switch the FMU to Event mode at the appropriate time. Such mechanisms are represented by $(s_m', h, b) = \texttt{stepT}_m(s_m, H)$, when $b = true$. In this case, s_m' represents the state of the FMU ready to begin the Event mode, at super-dense time $(t_R + h, 0)$, where t_R is the real part of the time of s_m. As discussed in Section 3.1, the causes of $b = true$ can be many.

Ticking Clocks. In Event mode, m in state $s_m \in S_m$ may activate any triggered output clock $y_m^c \in Y_m^c$, a fact that can be communicated to the importer via the function call $\texttt{get}_m^c(s_m, y_m^c)$. Conversely, any input clock $u_m^c \in U_m^c$ that needs to be ticked (according to the interval information), is activated by the importer, through the function call $\texttt{set}_m^c(s_m, u_m^c, true)$.

Enabling/Disabling Clock Equations. Let $s_m \in S_m$ denote the state of m right after a clock w_m^c (input or output) has been activated, and let (t_R, t_I) represent the current super-dense time. When w_m^c is activated, there is a set of equations, associated to w_m^c, that becomes active (are enabled) for the current super-dense time instant (t_R, t_I). The set of output variables whose value is computed by w_m^c's equations is denoted as "w_m^c's variables". While w_m^c is active, invocations to $\texttt{get}_m$ on w_m^c's variables will trigger their computation according to w_m^c's equations. However, the values that w_m^c's variables acquire while w_m^c is active are only made permanent when $\texttt{commit}_m^c$ is invoked. If $\texttt{set}_m^c$ is invoked to de-activate an active clock w_m^c at time (t_R, t_I) (before $\texttt{commit}_m^c$ is invoked), then m should ensure that w_m^c's variables return to the values they had immediately before w_m^c became active (this is not a strict requirement, since those variables should not be consulted once w_m^c became inactive). When $\texttt{stepE}_m$ is invoked, w_m^c becomes inactive along with its equations, and the super-dense time instant becomes $(t_R, t_I + 1)$. If $\texttt{stepE}_m$ is invoked before $\texttt{commit}_m^c$ is invoked, then $\texttt{commit}_m^c$ is invoked by m.

Propagating Clock Activations. In Event mode, if a clock w_m^c is (in-)active at super-dense time (t_R, t_I), then the importer must ensure that all other clocks that are connected to w_m^c must also be (in-)active for time (t_R, t_I). Triggered output clocks may activate during a super-dense time instant, or after a call to $\texttt{stepE}_m$. Therefore, the importer must query triggered output clocks to monitor activations. FMI allows declaring the variables in the XML file that may influence a triggered output clock (recall Figure 2). It is outside the scope of FMI to ensure that a simulation scenario is well defined (e.g., it does not result in an infinite number of (de-)activations).

Scheduling Time-Based Clocks. In Event mode, after a call to $\texttt{stepE}_m$, at super-dense time (t_R, t_I), m must be able to inform the importer of the time of the next tick of each clock $u_m^c \in U_m^c$ that is tunable, changing, or countdown. This is done through function $\texttt{nextT}_m$, when u_m^c satisfies one of the following conditions: 1. u_m^c is a countdown clock; or 2. u_m^c is not a countdown clock, and u_m^c was active in the super-dense time that was just concluded, at time $(t_R, t_I - 1)$. The Importer should use this information to schedule the next Event mode. If $\texttt{nextT}_m$ returns 0, then the importer must do a new event iteration.

Generic Clocked Simulation Algorithm. The following summarizes the Event Mode algorithm that coordinates the simulation with multiple FMU instances, with connected inputs/outputs and clocks. Let M denote the set of FMU instances participating in the simulation. We assume that one FMU instance $m \in M$ or the importer has requested to enter Event mode. Therefore we assume that every other instance $m' \in M \wedge m' \neq m$ has been stepped up to the same super-dense time $(t_R, 0)$. In the following, we use _ to denote an non-important argument.

1. Every $m \in M$ enters Event mode (super-dense time instant is $t_I = 0$);
2. Activate any time-based clocks scheduled to tick at $(t_R, 0)$, by invoking $\texttt{set}_{m'}^c(\_, w_{m'}^c)$ for any input/output clock $w_{m'}^c \in W_{m'}^c$ and any instance $m' \in M$;
3. Construct and solve system of equations for t_I:
   (a) For all $y_m^c \in Y_m^c$ of any instance $m \in M$, forward activation state of triggered clocks:
      i. Invoke $\texttt{get}_m^c(\_, y_m^c)$, and $\texttt{set}_{m'}^c(\_, u_{m'}^c)$ or $\texttt{get}_{m'}^c(\_, y_{m'}^c)$, for any other clock $u_{m'}^c \in U_{m'}^c$ or $y_{m'}^c \in Y_{m'}^c$ and instance $m' \in M$ that is transitively connected to y_m^c or has become active as a result of the clock activations;
      ii. Invoke $\texttt{commit}^c(\_, w^c)$ for any active input/output clock w^c whose input variables have been set.
   (b) Invoke $\texttt{get}_{m'}(\_, y_{m'})$ and $\texttt{set}_{m'}(\_, u_{m'}, \_)$ in the appropriate order, for any instance $m' \in M$.
4. Invoke $\texttt{stepE}_m(\_)$ for $m \in M$ (signals end of Event iteration t_I).
5. Schedule clocks by invoking $\texttt{nextT}_m$ on every relevant clock, for $m \in M$.
6. If any $m \in M$ wishes to repeat the event iteration, or if a clock returned a zero interval, go to Step 3 (start iteration $t_I + 1$).

The goal of Step 3 is to solve the system of equations that became active due to the clock activations. There are no guarantees that such a system has a solution, or that the clock activations will stabilize. It is up to the Importer to determine this, so we leave it intentionally unspecified.

4 Scheduled Execution

SE and SC have the following in common: they use the same clock types, as introduced in Section 2; directly connected clocks (e.g., y_m^c and u_n^c in Figure 2) will tick at the same simulated times (although the corresponding equations will be executed at different wall-clock times, see below); after a clock tick, there may be more clock ticks, either at the same time, or at some time in the future. However, there are differences, detailed later:

- Each SE clock w, when activated at simulated time $t_R \in \mathbb{R}_{\geq 0}$, represents a task that needs to be executed. In contrast, in SC, w merely enables a set of equations that are subsequently solved.
- In SE, there is a clear distinction between the wall-clock time, and the simulated time. For example, two clocks may tick at the same simulated time $t_R \in \mathbb{R}_{\geq 0}$ (because they are connected, or because they have the same period), but their corresponding tasks will execute at different wall-clock times. However, the two tasks will be computed with simulated time t_R.
- In SE, the execution of a task can be pre-empted by a higher priority task. This has the important consequence that the FMU must inform the importer of when a task should not be pre-empted.

The main goal of SE is to facilitate real-time simulation.

4.1 Motivating Example

Figure 6 shows an abstract example, where an FMU declares three input clocks and one output clock. Each input clock, when ticked, instructs the importer, who acts as a task scheduler (recall Figure 1), to execute the corresponding model partition (defined next) as soon as possible.

A model partition, or just partition, represents code that should be executed as soon as (in real time) an input clock ticks. Partitions contain arbitrary code that reads the inputs of the FMU, writes to the FMU's local variables (which can be shared among tasks) and outputs, and can trigger other clocks or update their interval. The inputs to each partition are set by the importer immediately before executing that partition, as part of the task corresponding to that partition. In Figure 6, u_m^c's partition reads and writes the shared variable x_m, and either updates the interval of v_m^c or ticks y_m^c.

We stress the distinction between model partition and a task: the former represents code that is executed within the context of the later. So a task T contains code that sets the inputs of the FMU, invokes the model partition P, and reads the outputs. Such a task will simply be denoted as "P's task". For example, in Figure 6, when execution Partition 1's task, the importer sets the values for input u_m before executing Partition 1.

In SE, there is a need for a function that the importer invokes, to tell the FMU to execute a partition. Note the difference between the partition activation function (defined later) and the clock set/get functions: the later inform the importer that a task should be scheduled, while the former executes as part of the previously scheduled task.

In Figure 6, input clock u^c_m ticks every 10ms and w^c_m ticks every 50ms, so every so often, the two clocks will tick simultaneously. When that happens, the scheduler needs to know whose task has the highest priority. As a result, the FMU needs to declare a priority level for each input clock. In Figure 6, u^c_m's task (the one executing Partition 1) should be executed before w^c_m's (Partition 3).

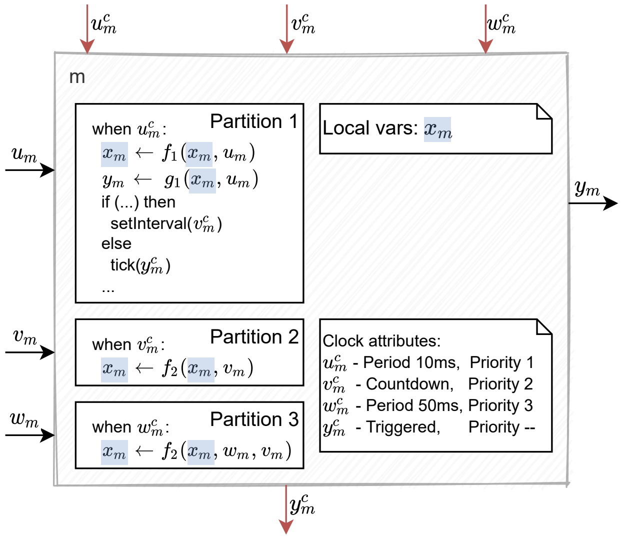

u^c_m
v^c_m
w^c_m
m
u_m
v_m
w_m
y_m
y^c_m
Partition 1
when u^c_m:
$x_m \leftarrow f_1(x_m, u_m)$
$y_m \leftarrow g_1(x_m, u_m)$
if (...) then
setInterval(v^c_m)
else
tick(y^c_m)
...
Local vars: x_m
Partition 2
when v^c_m:
$x_m \leftarrow f_2(x_m, v_m)$
Partition 3
when w^c_m:
$x_m \leftarrow f_2(x_m, w_m, v_m)$
Clock attributes:
u^c_m - Period 10ms, Priority 1
v^c_m - Countdown, Priority 2
w^c_m - Period 50ms, Priority 3
y^c_m - Triggered, Priority --

Figure 6. Motivating example, where an FMU declares three input clocks and one output clock.

Output clocks, in SE, are never directly associated to a partition of the FMU where they are declared. Instead, these can be connected to input clocks (including the ones of the owning FMU).

Because tasks can be pre-empted, certain operations, such as updating a shared variable, must be atomic (see example below). As such, the FMU must inform the importer of when it should not be interrupted, to prevent mixed resource access that would create inconsistent values.

Since partitions can trigger and update the interval of other clocks, there must be a mechanism for the FMU, in the middle of the calculation of a partition, to inform the importer that a clock has ticked or has a new interval, so that the importer can schedule the corresponding tasks.

Figure 7 illustrates a possible execution trace of the tasks corresponding to the partitions declared in Figure 6. At the initial wall-clock time, both task 1 and 3 are scheduled to execute. Since Task 1 has higher priority, it runs first, and Task 3 is delayed. While executing Task 1, the FMU informs the importer that v^c_m's task (Task 2) should be scheduled to run at wall-clock time t_2. At wall-clock time t_2, Task 1 is still executing, so Task 2 is delayed until wall clock time t_3. At t_3, Task 2 starts executing, but note that the activation time of Task 2 is still its scheduled time t_2. This is where the wall-clock time t_3 differs from the simulated time t_2. At t_4, Task 2 is pre-empted, because of Task 1. Finally, after being delayed substantially, Task 3 gets to execute, with its simulated time t_0.

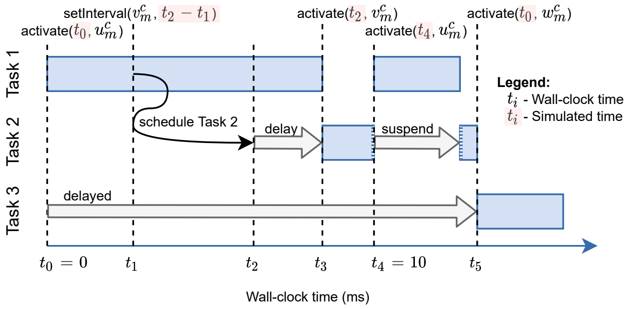


Figure 7. Example execution trace of Figure 6.

4.2 Scheduled Execution Semantics

The following formalization is a simplification meant to highlight the main functions defined in the FMI Standard. The main concepts being formalized are tasks, clocks, and activation of model partitions.

Definition 2 (SE FMU Instance)**.** An SE FMU instance with identifier m is represented by the tuple

$$\langle S_m, U_m, Y_m, U^c_m, Y^c_m, \\ \texttt{set}_m, \texttt{get}_m, \texttt{get}^c_m, \texttt{activate}_m, \texttt{nextT}_m \rangle$$

where:

- S_m, U_m, Y_m, U^c_m, and Y^c_m, are defined as in Definition 1.
- $\texttt{set}_m : S_m \times U_m \times \mathscr{V} \to S_m$ and $\texttt{get}_m : S_m \times Y_m \to \mathscr{V}$ are functions to set the inputs and get the outputs, respectively. In contrast with the SC FMU in Definition 1, $\texttt{get}_m$ does not alter m's state because any nontrivial computation of outputs should be done in the partitions associated with the input clocks, executed through the invocation of the $\texttt{activate}_m$ function.
- $\texttt{get}^c_m : S_m \times Y^c_m \to S_m \times \mathbb{B}$ queries the output clocks. Note that, in contrast to SC, $\texttt{get}^c_m(\_, y^c_m)$ changes the state of m, because it automatically de-activates y^c_m (the justification is provided below).
- $\texttt{activate}_m : S_m \times U^c_m \times \mathbb{R}_{\geq 0} \to S_m$ is a function representing the computation of a partition. If m is in state s_m at wall-clock time t_i, $s_m{}' = \texttt{activate}_m(s_m, u^c_m, t_i)$ represents three successive steps: the activation of clock u^c_m, the computation of the partition associated to clock u^c_m, and de-activation of clock u^c_m. In state $s_m{}'$, clock u^c_m will be inactive.
- $\texttt{nextT}_m : S_m \times U^c_m \to \mathbb{R}_{\geq 0} \cup \{NaN\}$ is the function that allows the importer to query the time of the next clock tick. It is defined as in Definition 1.

In addition, we will use the notation $\texttt{task}(u_m^c)$, for $u_m^c \in U_m^c$, to denote the task that will execute u_m^c's partition.

Scheduling Tasks. In SE, the importer operates as a scheduler of tasks that will activate the model partitions. As summarized in Table 1, clocks can be ticked by the FMU or importer, but we will focus on input clocks, since these are the ones that can be associated to a partition (when an output clock ticks, the importer is responsible for ticking all connected input clocks and therefore scheduling the corresponding tasks). Right after invoking $(s_m', true) = \texttt{get}_m^c(\_, y_m^c)$ on an output clock y_m^c that is active, the clock y_m^c should be inactive in state s_m'.

An input clock u_m^c may tick, and its period may be updated, in the middle of an $\texttt{activate}_m$ computation. Because $\texttt{task}(u_m^c)$ may be of high priority, the importer must not wait until the end of $\texttt{activate}_m$ to schedule $\texttt{task}(u_m^c)$. As such, the importer implements a function $\texttt{update}_m : S_m \to S_m$ defined by the FMI, that the FMU can invoke (in the FMI Standard, this is implemented as a callback mechanism). The importer, inside $\texttt{update}_m$, may consult the status of clocks and their intervals (through $\texttt{get}_m^c$ and $\texttt{nextT}_m$ functions), and schedule the corresponding tasks accordingly.

The time at which the importer schedules $\texttt{task}(u_m^c)$ is computed according to: u_m^c's declared interval; function $\texttt{nextT}_m$; or through the $\texttt{get}_{m'}^c(\_, y_{m'}^c)$ function of some other clock $y_{m'}^c$ and FMU instance m'. In the last case, $\texttt{task}(u_m^c)$ is scheduled to execute as soon as possible, according to the priorities known to the importer.

Executing Tasks. A task $\texttt{task}(u_m^c)$ that is scheduled to time t_i, due to the priorities chosen and consequent delays incurred, may only execute at a later wall-clock time $t_j > t_i$. When $\texttt{task}(u_m^c)$ is executed, it should set the relevant inputs through function $\texttt{set}_m$ (the importer knows the relevant inputs through the XML of m), activate the partition trough function $\texttt{activate}_m(\_, u_m^c, t_i)$, and possibly read the calculated outputs, through $\texttt{get}_m$.

Safeguarding Pre-emption. Unless otherwise stated by the FMU or importer, a task can be pre-empted at any moment. In order to allow the FMU to inform its environment that the currently executing task should not be pre-empted, the FMI defines two functions: `lockP` and `unlockP` that the FMU and importer can invoke, and are implemented by the importer. `lockP` informs the environment that a task cannot be pre-empted until `unlockP` is invoked.

Generic Scheduled Execution Algorithm. Let M denote a set of FMU instances, assumed to be initialized.

1. Schedule $\texttt{task}(u_m^c)$, for all $u_m^c \in U_m^c$ and all $m \in M$, if interval of u_m^c is constant fixed, or calculated;
2. When $\texttt{update}_m(\_)$ is invoked, do:
   (a) Lock pre-emption with `lockP`;
   (b) If $(\_, true) = \texttt{get}_m^c(\_, y_m^c)$, schedule $\texttt{task}(u_{m'}^c)$ for any clock $u_{m'}^c$ that is transitively connected to y_m^c.
   (c) Unlock pre-emption with `unlockP`;
3. Each task $\texttt{task}(u_m^c)$ is implemented as:
   (a) Set the inputs of m using $\texttt{set}_m$ (locking pre-emption with `lockP` and `unlockP` if needed);
   (b) Invoke $\texttt{activate}_m(\_, u_m^c, t_i)$, where t_i is the simulated time that $\texttt{task}(u_m^c)$ was scheduled to execute.
   (c) Get the outputs of m using $\texttt{get}_m$ (locking pre-emption with `lockP` and `unlockP` if needed);

5 Related Work

Synchronous clocks are one of the solutions proposed to tackle the more general challenge of co-simulating hybrid systems. Other proposals have been made in the state of the art, but none of them tackle the problem of discerning different simultaneous events in the context of co-simulation. For instance, Cremona et al. (2016) proposes a master algorithm for hybrid co-simulation. The proposal includes support for absent signals, mandatory implementation of rollback, zero duration step size, co-simulation FMUs supporting feed-through, and predictable step sizes. However, it excludes algebraic loops, due to the introduced non-determinism. Our proposed interfaces enables algebraic loop resolution, even when clocks are involved, but does not provide guarantees of convergence.

An extensive study of hybrid system simulation challenges was carried out in Mosterman and Biswas (2000), and includes, for example, the possibility of an event iteration driving the system into chattering. And Tripakis and Broman (2014), Broman et al. (2015) and Liboni et al. (2018) focus such discussion in the context of the FMI Standard, providing solutions to some of these challenges. These works complement ours by helping importers assess whether a given simulation scenario is well behaved. We refer the reader to Gomes et al. (2018) for more references in co-simulation of hybrid systems.

The goal of this paper is to describe the main mechanisms standardized in the FMI Standard that enable synchronous clocked simulation and scheduled execution. We can therefore highlight related work that share the same goals.

Regarding SC simulation, we highlight the work in Otter, Thiele and Elmqvist (2012) and Elmqvist, Otter and Mattsson (2012), that introduce the synchronous clocks constructs used in the Modelica language, specified in Modelica Association (2021). Such work, and references thereof on synchronous languages (Benveniste et al. 2003; Colaço and Pouzet 2003), were used as basis for the definition of the SC approach described here. The main difference is that an SC clock does not enforce a partition on the equations that can be written by it. These differences make it more difficult to ensure well-formedness of co-simulation scenarios, but provide more flexibility, reflecting the heterogeneous use cases of FMI.

In the domain of scheduled execution, we highlight the OSEK/VDX (ISO 17356-3:2005 2005) and AUTOSAR

Standards, which enable different suppliers to develop and test software independently, and subsequently integrated the different applications. Such work complements the SE Interface by standardizing the importer environment, where FMU SE instances can execute.

6 Conclusion

This paper summarizes the results of the FMI project developing interfaces to interact with clocks. This is a challenging task, because the kinds of simulation scenarios covered can combine traditional events with clock ticks, and may possibly be ill-defined while still conforming to the FMI Standard. This is intentional, as the FMI aims at flexibility, placing the burden of ensuring well-formedness on the importer.

We have presented two interpretations of clocks. The main differences between them lie in the degree of control that the importer has over the duration of computations, and on the behavior of the independent variable with respect to the wall clock time. The formalization provided is meant as an introduction to the clocks and their conceptualization in the FMI Standard. The FMI Standard document is continuously being improved, and therefore remains the source of truth. We refer the reader to (Junghanns et al. 2021) for an account of the most important features being developed for the FMI 3.0.

Acknowledgements

Cláudio Gomes is grateful to the Poul Due Jensen Foundation, which has supported the establishment of a new Centre for Digital Twin Technology at Aarhus University.

We are thankful to Simon Thrane Hansen, Peter Gorm Larsen, and Casper Thule, for their helpful comments on this paper.

Finally, we are thankful to the Modelica association for the continued support to the FMI Project.

References

2.0, FMI (2014). *Functional Mock-up Interface for Model Exchange and Co-Simulation*. URL: https://fmi-standard.org/downloads/ (visited on 2019-09-15).

Benveniste, A. et al. (2003-01). “The Synchronous Languages 12 Years Later”. In: *Proceedings of the IEEE* 91.1, pp. 64–83. ISSN: 0018-9219. DOI: 10.1109/JPROC.2002.805826.

Broman, David et al. (2015). “Requirements for Hybrid Cosimulation Standards”. In: *18th International Conference on Hybrid Systems: Computation and Control*. HSCC ’15. Seattle, Washington: ACM New York, NY, USA, pp. 179–188. ISBN: 978-1-4503-3433-4. DOI: 10.1145/2728606.2728629.

Colaço, Jean-Louis and Marc Pouzet (2003). “Clocks as First Class Abstract Types”. In: *Embedded Software*. Ed. by Rajeev Alur and Insup Lee. Red. by Gerhard Goos, Juris Hartmanis and Jan van Leeuwen. Vol. 2855. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 134–155. ISBN: 978-3-540-20223-3 978-3-540-45212-6. DOI: 10.1007/978-3-540-45212-6_10.

Cremona, Fabio et al. (2016-11). “Step Revision in Hybrid Co-Simulation with FMI”. In: *14th ACM-IEEE International Conference on Formal Methods and Models for System Design*. Kanpur, India: IEEE.

Elmqvist, Hilding, Martin Otter and Sven Erik Mattsson (2012). “Fundamentals of Synchronous Control in Modelica”. In: *9th International Modelica Conference*. URL: https://elib.dlr.de/78420/.

Gomes, Cláudio et al. (2018). “Co-Simulation: A Survey”. In: *ACM Computing Surveys* 51.3, 49:1–49:33. DOI: 10.1145/3179993.

ISO 17356-3:2005 (2005). *Road Vehicles — Open Interface for Embedded Automotive Applications — Part 3: OSEK/VDX Operating System (OS)*. ISO 17356-3:2005, p. 61. URL: https://www.iso.org/standard/40079.html.

Junghanns, Cláudio et al. (2021). “The FMI 3.0 Standard Interface for Clocked and Scheduled Simulations”. In: *Proceedings of the 14th International Modelica Conference*. 14th International Modelica Conference. online: Linköping University Electronic Press, Linköpings Universitet, to be published. URL: https://ep.liu.se/en/conference-article.aspx?series=ecp&issue=169&Article_No=16.

Kübler, R. and W. Schiehlen (2000). “Two Methods of Simulator Coupling”. In: *Mathematical and Computer Modelling of Dynamical Systems* 6.2, pp. 93–113. ISSN: 1387-3954. DOI: 10.1076/1387-3954(200006)6:2;1-M;FT093.

Lee, Edward A. and Haiyang Zheng (2005). “Operational Semantics of Hybrid Systems”. In: *Hybrid Systems: Computation and Control*. Vol. 3414. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 25–53. ISBN: 978-3-540-25108-8. DOI: 10.1007/978-3-540-31954-2_2. pmid: 20646090.

Liboni, Giovanni et al. (2018-01). “Beyond Time-Triggered Co-Simulation of Cyber-Physical Systems for Performance and Accuracy Improvements”. In: *10th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*. Manchester, United Kingdom. URL: https://hal.inria.fr/hal-01675396.

Modelica Association (2021). *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling*. Language Specification Version 3.5. Modelica Association.

Mosterman, Pieter J and Gautam Biswas (2000-08). “A Comprehensive Methodology for Building Hybrid Models of Physical Systems”. In: *Artificial Intelligence* 121.1–2, pp. 171–209. ISSN: 0004-3702. DOI: http://dx.doi.org/10.1016/S0004-3702(00)00032-1.

Otter, Martin, Bernhard Thiele and Hilding Elmqvist (2012). “A Library for Synchronous Control Systems in Modelica”. In: *9th International Modelica Conference*. URL: https://elib.dlr.de/79763/.

Tarjan, Robert (1971-10). “Depth-First Search and Linear Graph Algorithms”. In: *12th Annual Symposium on Switching and Automata Theory (Swat 1971)*. Vol. 1. 2. East Lansing, MI, USA. ISBN: SMJCAT000001000002000146000001. DOI: 10.1109/SWAT.1971.10.

Tripakis, Stavros and David Broman (2014). *Bridging the Semantic Gap Between Heterogeneous Modeling Formalisms and FMI*.

Engineering Domain Interoperability Using the System Structure and Parameterization (SSP) Standard

Robert Hällqvist[1,4] Raghu Chaitanya Munjulury[2,4] Robert Braun[4] Magnus Eek[3] Petter Krus[4]

[1]System Simulation and Concept Development, Saab Aeronautics, Sweden, robert.hallqvist@saabgroup.com
[2]Technical Management & Maintenance, Saab Aeronautics, Sweden
[3]Technology & Innovation Management, Saab Aeronautics, Sweden
[4]Division of Fluid and Mechatronic Systems (FLUMES), Linköping University, Sweden

Abstract

Establishing interoperability is an essential aspect of the often-pursued shift towards Model-Based System Engineering (MBSE) of, for example, aircraft. If models are to be the primary information carriers during development, the applied methods to enable interaction between engineering domains need to be modular, reusable, and scalable. Given the long life cycles and often large and heterogeneous development organizations in the aircraft industry, one possible solution is to rely on open standards and tools. In this paper, the standards Functional Mock-up Interface (FMI) and System Structure and Parameterization (SSP) are exploited to exchange data between the disciplines of systems simulation and geometry modeling. A method to export data from the 3D Computer Aided Design (CAD) Software (SW) *CATIA* in the SSP format is developed and presented. Analogously, FMI support of the Modeling & Simulation (M&S) tools *OMSimulator*, *OpenModelica*, and *Dymola* is utilized along with the SSP support of OMSimulator. The developed technology is put into context by means of integration with the M&S methodology for aircraft vehicle system development deployed at Saab Aeronautics. Finally, the established interoperability is demonstrated in an industrially relevant use-case. A primary goal of the research is to prototype and demonstrate functionality, enabled by the SSP and FMI standards, that could improve on MBSE methodology implemented in industry and academia.

Keywords: FMI, SSP, Modeling and Simulation, CATIA, OMSimulator, OpenModelica, Dymola

1 Introduction and Motivation

Each engineering domain has its preferred methods and tools for design and analysis, implying that different but often overlapping views of one single system are modeled using several different tools. For some applications, managing all views of interest in a tool suite provided by a single tool vendor may be possible, but for other applications, this may be neither feasible nor desirable. Exchanging information between engineering domains using tool-to-tool connections introduces a set of unwanted drawbacks that may increase in significance with increased product life cycle length. Such connections are fragile and may require significant maintenance. This motivates the utilization of standards to ensure continuity and consistency in the digital thread.

Traditionally at Saab, in-house standardized interface formats are used for the manual exchange of data between different engineering domains. Even though the applied MBSE methods are generally considered successful, such processes are error-prone, tedious, and difficult to maintain; particularly considering the previously mentioned long life cycles of aircraft and aircraft sub-systems. As a result, there is a risk that data may be exchanged less frequently than it should be. This may be a limiting factor in M&S credibility and the risk of taking sub-optimal model-based design decisions is increased as a consequence.

At Saab Aeronautics, aircraft vehicle system models are developed according to the *Saab Aeronautics Handbook for Development of Simulation Models* (Andersson and Carlsson 2012). This handbook highlights aspects such as the definition of model specifications, intended use(s), and the importance of conducting Verification & Validation (V&V). These activities are all highlighted as essential in the literature (Roza, Voogd, and Sebalj 2012; Roy and Oberkampf 2011; International Council on Systems Engineering 2015).

The process described in the handbook can largely be described by the workflow visualized as the *Sub-system Model* abstraction level of Figure 1. Furthermore, the cornerstones of *Sub-system Model* development are analogous to *Simulator* and *Component Model* development if viewed at the level of detail presented in Figure 1. The artifacts at each level of abstraction are executable simulation models, or simulators, of the mathematical sort (Ljung and Glad 2004; Peter Fritzson 2004).

The SSP (Modelica Association 2019) and FMI (FMI Development Group 2020) standards provide standardized formats for establishing interoperability between the different levels of abstraction. The FMI standard concern the export of models for integration in *simulator* (Hällqvist 2019) applications and the SSP standard primarily focuses on the definition and export of simulators for integration into simulator applications at a higher level of abstraction or for exploitation in different frames of reference.

DOI
10.3384/ecp2118137

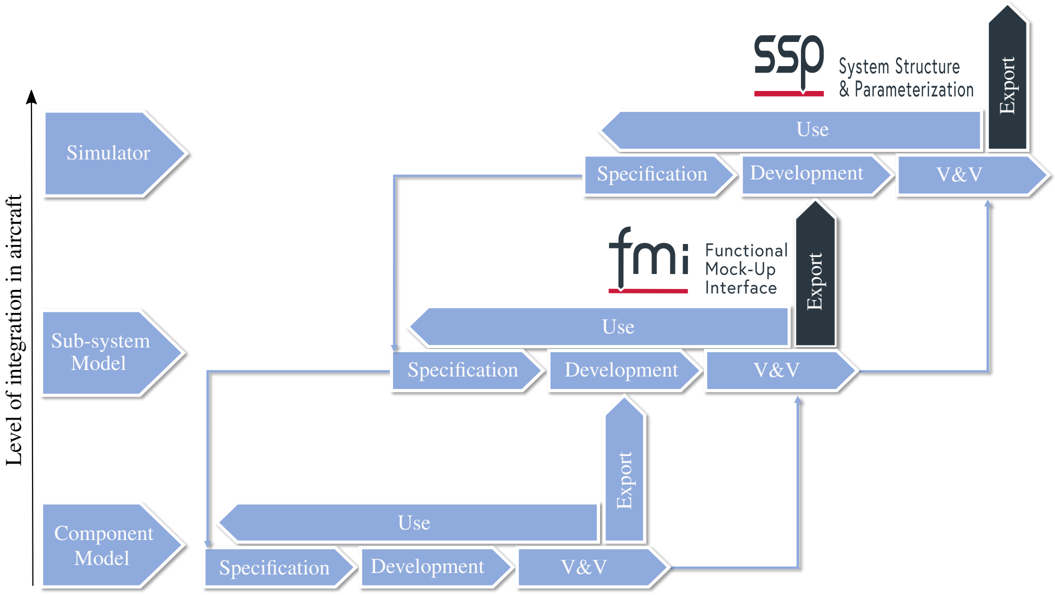


Figure 1. Simulation application development process connecting the Component Model, Sub-system Model, and Simulator levels of abstraction. The term *Model* here refers to mathematical simulation models as specified by Ljung (Ljung and Glad 2004) and Fritzon in (Peter Fritzson 2004). A simulator here is also seen as a mathematical simulation model composed of several connected models or simulators exported from a lower level of hierarchy (Hällqvist 2019). The V&V activities are seen as bottom up, a view that is in line with the *Validation Level per Level* approach as described by (International Council on Systems Engineering 2015). The natural connection between the two standards, FMI and SSP, and the activities of the simulation application development process are highlighted in the figure.

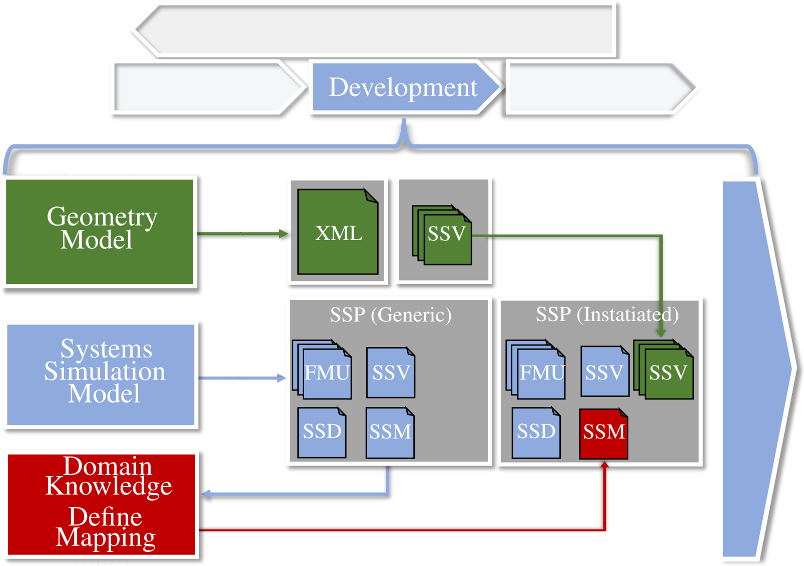


Figure 2. Expansion of the development activity specified in Figure 1. The workflow is designed to be applicable to the three system levels shown in Figure 1: the Component Model, Sub-System Model, and Simulator levels of abstraction. The colors in the figure indicate the source of the information. The rightmost SSP in the figure, *SSP (Instantiated)*, conforms to a executable specified using information from all the relevant engineering domains.

Additionally, the SSP standard provides a standardized format for specifying the parameters of M&S artifacts representing more than a single realized aircraft system, subsystem, or component. The approach taken here is to exploit these parts of the standard, at all of the presented levels of abstraction, to enable a tool agnostic method to exchange information concerning geometrical parameters and their bindings to the corresponding mathematical simulation models and simulators. As a consequence, Figure 1 is complemented by Figure 2 which is described in detail in Section 4.

This paper is structured as follows. In Section 2, the enablers of the work are introduced and related to the presented research. Furthermore, geometry modeling is related to the targeted standards in Section 3. The proposed refinement of the existing M&S application development methodology implemented at Saab is described in Section 4. In Section 5, the relevant details of the selected application example are described; this application example is then subjected to a use-case presented in Section 6. The use-case results are presented and discussed in Section 7. Finally, the conclusions of the work are stated in Section 8.

2 Interoperability via open tools and standards

Standardized and automated connections of CAD to other interdependent engineering domains is by no means a new research area and a plethora of solutions have been proposed in the literature, including published research led by Saab (Lind and Oprea 2012).

In 1999, Engelson et al. formulated a method to transform mechanical domain *SolidWorks* geometry models into the format of the standardized multi-domain M&S language Modelica (Engelson, Larsson, and P. Fritzson 1999). Similarly, Elmqvist et al. developed a tool for the automatic translation of CATIA multibody models into Modelica code (Elmqvist, Mattsson, and Chapuis 2009). Remond et al. employ a similar approach to generating Modelica models of thermo-fluid piping networks based on CAD data from CATIA (Remond, Gengler, and Chapuis 2015).

Furthermore, Baumgartner et al. developed *Dymola* functionality for generating Modelica models from CATIA multibody models. Their approach explicitly includes the storing of parameter values in a separate text format such that the geometrical data can be modified and updated without re-generating the complete model or package (Baumgartner and Pfeiffer 2014). This conscious separation is of particular interest during the development of aircraft models because such models and libraries often have long life cycles spanning a significant period of the aircraft's development and operation.

These publications all primarily focus on exchanging or generating complete executable models detailed with the geometry model information expressed in the CAD tool of the author's choice. However, during development, and later life cycle stages, the overall structure of the models is less prone to change. A component may be enhanced, the dimensions of a pipe may be modified, but what components are used and what parameters need to be exchanged is likely to be well established information. Lind et al. present such a use-case where the modeled geometry of aircraft fuel tanks is exploited in order to automatically increase the fidelity of the corresponding Modelica models by means of fitting Radial Basis Functions (RBFs) networks to the extracted data (Lind and Oprea 2012).

Even though it is similar, the focus of the research presented here is instead on only exchanging model parameters. That way, the domain specific engineering tools can be used as originally intended but with relevant, and up-to-date, information from the application specific relevant neighboring engineering domains. The SSP standard format is identified as an applicable solution to the above mentioned problem.

2.1 System Structure and Parameterization (SSP)

Three of the SSP Extensible Markup Language (XML) formats are the focal point of the research presented here: the System Structure Description (SSD), System Structure Parameter Values (SSV), and System Structure Parameter Mapping (SSM). The SSD is an XML file primarily specified to describe the architecture of a set of coupled mathematical models, henceforth referred to as a simulator. In the SSV file, it is possible to specify the values of the parameters of the simulator constituent model's. However, the SSV file does not necessarily define the mapping between the parameter values and the parameter names. These bindings can instead be specified in a SSM file. Within this approach, the SSM file is specified once for each constituent model version; changes only need to be made if the parameter interface is modified. The SSV files are changed as soon as the parameter values are modified. A highly parametric model can represent many different physical systems, sub-systems, or components using various SSV input files.

2.2 Use, exchange, and manipulation of SSPs

The FMI and SSP standards are both utilized together with *Transmission Line Modeling (TLM)* (Auslander 1968; Krus et al. 1990) in the simulation environment OMSimulator. The OMSimulator is an open-source master simulation tool originally developed during the ITEA3 project *OpenCPS* (OpenCPS Project Partners 2019). It is based on a simulation framework developed by the Swedish bearing manufacturer SKF for connecting models of bearings with models from external tools (Peter Fritzson et al. 2020; Ochel et al. 2019). The OMSimulator is a stand-alone M&S tool maintained by the Open Source Modelica Consortium (OSMC). It is available as a plugin to the OMEdit Graphical modeling tool which enables graphical, and textual, development of simulators. Other tools with similar support are available; however, they are few in number because the SSP standard is still young. A list of the tools officially supporting the SSP standard is provided at the SSP project web page (Modelica Association Project System Structure and Parameterization 2021).

The OMSimulator is used as an integrating simulation tool as it supports both of the two targeted standards. Being open-source, it is also used as a platform for implementing the necessary prototype SSP manipulation functionality that is required for the approach described in Section 4.

OMSimulator functionality, using either the available *Lua* or *Python* Application Programming Interface (API) to export template SSM and SSV files is available. A subset of the OMSimulator Python API specifically relevant to the presented research is provided in Listing 1.

Listing 1. OMSimulator Python API commands particularly relevant for the presented research

```
1) oms.exportSSMTemplate("<submodel>.fmu",
"<submodel>.ssm")
2) oms.exportSSVTemplate("<submodel>.fmu",
"<submodel>.ssm")
3) oms.export("<model>", "<model>.ssp")
4) oms.exportSnapshot("<model>")
5) oms.importFile( "<model>.ssp")
```

The API command `oms.exportSSMTemplate` triggers the export of all signals in the *<submodel>.fmu*, that have a start attribute, to an SSM file. The extract of an example SSM file generated using the API command is presented in Listing 2. One *MappingEntry* is generated for each parameter. Each *MappingEntry* has a target and a source attribute. The target is mapped to the name of each *<submodel>.fmu* parameter and the source is left unspecified. The source attribute allows for the manual specification of the mapping to the corresponding parameter value in an SSV file. If a source in the generated SSM is left unspecified, then the *<submodel>.fmu* parameter is left at its default value as presented in the Functional Mock-up Unit (FMU) *<ModelDescription>.xml* file.

Listing 2. Example of SSM file template generated using the OMSimulator API command `oms.exportSSMTemplate`

```
<ssm:ParameterMapping xmlns:ssc="http://ssp-
    standard.org/SSP1/SystemStructureCommon"
    xmlns:ssm="http://ssp-standard.org/SSP1/
    SystemStructureParameterMapping" version="
    1.0">
<ssm:MappingEntry source="" target="<submodel.
    parameterA>"/>
</ssm:ParameterMapping>
```

The API command `oms.exportSSVTemplate` enables the default parameter values of the *<submodel>.fmu* to be exported to an SSV file. An extract of an example SSV file generated using the API command is presented in Listing 3. A *Parameter* entry is generated for every signal with a start attribute in the FMU *<ModelDescription>.xml* file. Each parameter entry has a *name* attribute corresponding to the *<submodel>.fmu* parameter name, a *unit* attribute, and a *value* attribute corresponding to the default parameter value, i.e. the value of the *<ModelDescription>.xml* start attribute. The SSV file parameters are mapped via name matching to the FMU parameters if a SSM file is unavailable.

Listing 3. Example of SSV file template generated using the OMSimulator API command `oms.exportSSVTemplate`

```
<ssv:ParameterSet name="
    modelDescriptionStartValues">
 <ssv:Parameters>
  <ssv:Parameter name="<submodel.parameterA>">
   <ssv:Real value="<submodel>.<ModelDescription
       >.parameterA.value>" />
  </ssv:Parameter>
 </ssv:Parameters>
</ssv:ParameterSet>
```

Additionally, In Listing 1, the Python API command for exporting an SSD description of the architecture (`oms.exportSnapshot`) is presented, along with the command for exporting, and importing, a complete SSP package (`oms.export`). An extract of an example SSD file generated using either of the two export APIs is presented in Listing 4. The SSD extract visualizes how the SSV and SSM files are incorporated into the model or simulator architecture. The *ssd:ParameterBinding* element has a *source="resources/<values>.ssv"* attribute pointing to the SSV file used. This element also has a child that, again via the *source="resources/<bindings>.ssm"* attribute, specifies the SSM file used. A complete description of all the available OMSimulator API functions is provided in the user manual (Ochel 2021).

Listing 4. Example of SSD file generated using the OMSimulator API command `oms.exportSnapshot`. The SSD file connects the FMU parameters to the parameter values in the SSV file via the mappings expressed in the SSM file.

```
<ssd:SystemStructureDescription
 <ssd:System name="root">
  <ssd:ParameterBindings>
             <ssd:ParameterBinding source="
                 resources/<values>.ssv">
                 <ssd:ParameterMapping source="
                     resources/<bindings>.ssm"/>
             </ssd:ParameterBinding>
       </ssd:ParameterBindings>
    </ssd:System>
</ssd:SystemStructureDescription>
```

3 Geometry modeling and SSP

At Saab Aeronautics, geometry models are developed according to the *Knowledge-Based Engineering (KBE)* methodology MOKA (Stokes 2001). MOKA specifies an iterative process whereby the developed models can be added or updated in steps at each stage of the methodology. The geometry models created are parametric in nature. For example, the sizes of all the components in the application example coolant distribution system can be modified alongside any changes in the specification. The *User Defined Features (UDF)* created in CATIA encapsulate the design intent and design automation is applied

DOI
10.3384/ecp2118137

to instantiate the pipes based on the input points. All the parameters needed for simulation are computed automatically because the knowledge/calculations are embedded in the UDF.

Inspired by the previous work conducted at Saab and Linköping University (Lind and Oprea 2012; Munjulury et al. 2016; Munjulury 2017), Visual Basic for Applications (VBA) is exploited to extract parameter information from CATIA via a set of macros. Additionally, the implemented macros convert the extracted information to the SSV format.VBA macros can either be executed directly from CATIA or via, for example, a User Interface (UI) in Microsoft Excel. Application-specific mapping, as described in Section 3.2, is applied to the intermediate XML.

3.1 Parameter extraction and SSP support

An approach similar to that of Munjulury et. al. (Munjulury et al. 2016), exploiting the Document Object Model (DOM) objects available in VBA, is used to create an intermediate CATIA XML. This intermediate CATIA XML conversion is tailored to reduce the number of data interfaces enabling a robust and seamless exchange to other formats, such as SSV. For every entity (point, line, plane, surface, etc.,) at least three parameters are created when designing a component in CATIA. The functionality to extract and save all the geometry model parameter values into an XML file is available in, for example, *3D experience*; however, it is all the more challenging to find, extract, and store specific parameters. First of all, the latter requires a list of identifiers. Identifiers in this scenario are the *Parameter sets* or *Geometrical sets* that contain the parameters in the respective identifier list to help narrow down the search. The following are the steps involved in creating the SSV file.

- The user provides the Identifiers in the UI in order to narrow down the total number of parameters that need to be saved to the intermediate CATIA XML.
- A first VBA macro is executed which reads all the parameters available in the CATIA geometry model. The parameters specified in the identifier list mentioned above are then extracted from the geometry model and saved in the CATIA XML. The CATIA XML structure maps to the CATIA Product tree structure.
- A second VBA macro converts the CATIA XML to an SSV file.

3.2 Integration of application specific functionality

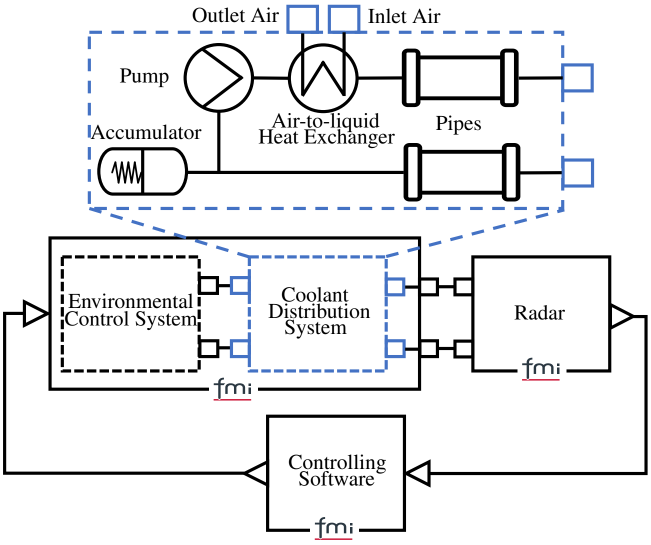
Functionality enabling application specific mappings has been developed. This functionality is kept separate from the SSP support macros such that aggregation methods can be exchanged and tailored to different applications. The methodology to instantiate pipes and insulation using the UDF is an add on to the methodology currently used at Saab Aeronautics to create the respective components. This add on reduces the time needed for the design process as most of the process is successfully automated; the only additional modeling requirement is to include the bend points of the pipes. With this automation, the parameters needed by the mathematical models of the application example are created and recursively used to compute the aggregated parameter values needed for the lumped parameters. The developed application specific functionality is,

- The combination of parameter values, such as fluid pipe lengths, enabling lumped parameter dynamic simulation components
- Unit transformations,
- Interpolating in application specific interpolation tables used to, for example, convert bends in pipes to pressure loss coefficients.

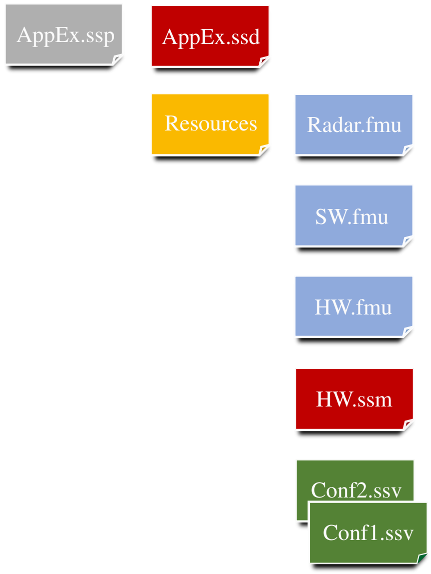
4 Proposed concept

This section describes the proposed methodology for exploiting the established connection between the domains of geometrical modeling to that of mathematical modeling and system simulation. The methodology is here related to the general simulation application development process presented in Figure 1. In Figure 2, the proposed additions to the *Development* activity of Figure 1 are described in detail. The process visualizes, see the light blue artifacts in the figure, how the simulation application is first exported in the SSP format, including an SSD architecture description, an SSV file containing default parameter values of the parameterized simulator, sub-system model, or component model, and a template SSM file containing empty bindings to the aforementioned parameters. In parallel, the parameter values of the corresponding geometry model are expressed in the SSV format, visualized as green artifacts in the figure. In the next step, the SSM file is populated with the geometry model names of the corresponding simulation application parameters such that the geometry model SSV file is specified as the source of the parameter values. Please note that the process of Figure 2 needs to be iterated. However, the user input specified SSM file only needs to be updated if the parameter interface is modified. The rightmost SSP in the figure corresponds to a set of fully specified executable entities ready for V&V, as-is use, and integration into a simulation application at a higher level of abstraction.

The process of Figure 2 is described as applicable for the development activities at all of the levels of abstraction presented in Figure 1. However, the work here is primarily focused on the *Sub-system Model* and *Simulator* levels. The available tool support of the SSP standard is primarily focused on FMI applications. If parameters in components, present in for example Modelica component libraries, are to be specified using the SSP standard, then tool support for this type of functionality, in the relevant simulation application modeling tools, could render a more efficient exchange of information.


Outlet Air
Inlet Air
Pump
Accumulator
Air-to-liquid Heat Exchanger
Pipes
Environmental Control System
Coolant Distribution System
Radar
Controlling Software
fmi
fmi
fmi

(a) Application example schematic description. The dashed *Coolant Distribution System* is highlighted as its parameters are specified in the geometry modeling domain and exchanged using the concepts of the SSP standard. The individual parts of the application example are exported using the FMI standard.



(b) Application example SSP file structure. The SSP file represents two different simulator geometrical configurations via the two different incorporated SSV files *Conf1.ssv* and *Conf2.ssv*. The simulator architecture is described in the *AppEx.ssd* file.

Figure 3. Application example description. A schematic description of the application example architecture and its constituent executable models is provided in Figure 3a. The structure of the resulting application example SSP file is provided in Figure 3b.

5 Application example

The application example incorporates the targeted engineering domains and M&S tools; a schematic overview is provided in Figure 3. The application example is separated into three different main parts; each of these parts is exported as an FMU from its original development tool. The modeled *Coolant Distribution System* is highlighted in as dashed and blue in Figure 3a as it here is the target of the established interoperability between systems simulation and geometry modeling.

The FMUs of the application example are packaged in a SSP file providing a complete and executable description of the targeted simulator configurations, see Figure 3b. The depicted SSP file includes two different geometrical configurations. These configurations are specified through the two included SSV files *Conf1.ssv* and *Conf2.ssv*.

5.1 Mathematical modeling

Two of the three constituent parts of the application example are individual models of the mathematical sort. These two mathematical models include an interpolation based representation of a Environmental Control System (ECS), a liquid coolant distribution system, and a consumer of generated cooling power. The first two modeled coupled sub-systems, the ECS and the liquid coolant distribution system, are lumped together in a single exported FMU, whereas the consumer is separated from the other two, see Figure 3. The development of these aircraft sub-systems is typically conducted by different departments at Saab, or by suppliers, and this partitioning is intended reflect a likely situation during development.

Even so, all three modeled sub-systems are developed using components from the Saab Aeronautics in house Modelica library *Modelica Fluid Light* (Eek, Gavel, and Ölvander 2017) and the *Modelica Standard Library* (The Modelica Association 2019). The mathematical modeling is conducted in the Dymola and OpenModelica SW.

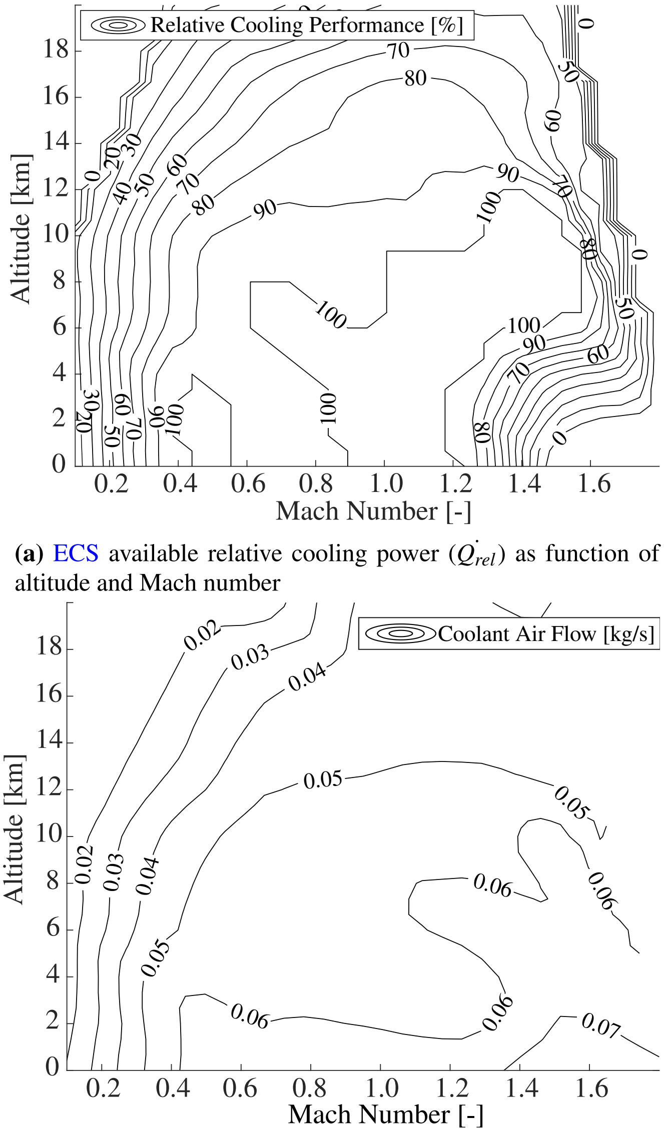
5.1.1 Environmental Control System (ECS)

A simulator enabling detailed studies of pilot thermal comfort was presented in (Hällqvist et al. 2018). The ECS presented here is based on the aircraft cooling system of that simulator. The ECS model incorporated in the application example is intended to provide a connection between the operating conditions and the cooling power available for distribution to the included consumer.

The results of maximum available steady-state relative cooling power $\dot{Q}_{rel}$, along with the corresponding available mass flow of conditioned cold air $\dot{m}$, are exploited in an interpolation based Modelica component, see Figure 4a and Figure 4b. Provided the characteristics of Figure 4, the minimum possible cold air temperature can be calculated as

$$T_{in}^{min} = T_{out} - \frac{\dot{Q}_{max}^{current}}{\dot{m} \cdot C_p} \tag{1}$$

where T_{out} is the current exhaust air temperature and $\dot{Q}_{max}^{current} = \dot{Q}_{rel} \cdot \dot{Q}_{max}$. The parameter $\dot{Q}_{max}$ specifies the maximum available cooling power independent of the operating conditions. The specific heat at constant pressure is denoted C_p in Equation 1. The ECS is here modeled



(a) ECS available relative cooling power ($\dot{Q}_{rel}$) as function of altitude and Mach number

(b) ECS available coolant flow ($\dot{m}$) as function of altitude and Mach number.

Figure 4. Steady-state characteristics of the application example ECS

as a source with a prescribed mass flow corresponding to that of $\dot{m}$. The temperature of the air expelled from the source is regulated by the incorporated control system, see Figure 3; however, a lower bound corresponding to that of T_{in}^{min} is specified in the ECS model.

5.1.2 Coolant distribution system

The application example's coolant distribution system is schematically described as the dashed and highlighted model in Figure 3. The cooling distribution system interfaces the modeled ECS via the Air-to-Liquid Heat Exchanger (LHEX). The LHEX transfers heat from the liquid circulated by the included pump to the ECS coolant air.

The Modelica components with parameters specified by the geometry model are all part of the *Modelica Fluid Light* library. The considered modeled components are a pipe, a pump, an LHEX, and an accumulator. The parameters of each modeled library component, here specified via the application example's geometry model, are presented in Table 1. A sub-set of the included model component equations are described in detail in the following paragraphs, in order to highlight how the selected parameters impact upon the characteristics of the coolant distribution system model.

Pipe	D_h	l	z
Accumulator	V_{acc}		
LHEX	h	b	l

Table 1. Summary of the parameters that are exchanged in the application example. The pipe parameters D_h and z represent the pipe hydraulic diameter and its lumped pressure loss coefficient. The pressure loss coefficient is a result of pipe bends and contractions/expansions. The parameter l represents pipe or LHEX length. The LHEX height and width are denoted h and b and the accumulator volume V_{acc}.

The pipe model algebraic equation relating the component parameters to mass flow ($\dot{m}$) and pressure drop (Δp) is

$$\Delta p = \frac{(z + c \cdot l/D_h)}{A^2 \cdot 2\rho}\dot{m}^2 \tag{2}$$

where A is the pipe's cross sectional area, D_h is the hydraulic diameter, and l is the pipe length (Miller 1990). The friction coefficient is denoted by c. The one-time pressure losses occurring as a result of pipe bends and contractions/expansions are incorporated via the parameter z. The pipe component parameters of Table 1 also impact upon the relationship between the air temperature surrounding the pipe and the specific enthalpy of the fluid itself. This relationship is described by a system of differential and algebraic equations:

$$\begin{aligned} q_{ep} &= A_o h_{ep}(T_e - T_p), \\ q_{pf} &= A h_{pf}(T_p - T_f), \\ \dot{T}_p &= (q_{ep} + q_{pf})/(C_p \cdot M), \\ \dot{h} &= \dot{m}/(\rho \cdot V)(h_{in} - h), \end{aligned} \tag{3}$$

where the pipe hull outer surface area is $A_o = \pi D_o L$ and the pipe hull inner surface area is $A = \pi D_h L$. The specific enthalphy h is modeled as a nonlinear function of the fluid temperature T. The pipe input specific enthalpy is denoted by h_{in}. The variable T_e represents the temperature of the media surrounding the pipe's outer surface whereas the variable T_p represents the pipe's hull temperature. The intermediate variables q_{ep} and q_{pf} represent the heat transferred from the pipe's external environment to its hull and from the pipe's hull to the fluid, respectively.

The modeled accumulator component exploits the Modelica inner/outer concept to access temperature dependent information concerning the system's total volume. The accumulator pressure is then related to the fluid temperature via linear interpolation as

$$p = \frac{p_f - p_e}{V_{acc}}\left[\sum_{i=1}^{N}(V_i(T) - V_i(T_0)) + V_{acc}(T_0)\right] + p_e \tag{4}$$

where $V_i(x)$ is the fluid volume in connected component model i, at the current temperature $x = T$ or the temperature at filling $x = T_0$. The pressure in the accumulator when it is full and empty are denoted p_f and p_e respectively. The two accumulator parameters V_{acc} and $V_{acc}(T_0)$ represent the total accumulator volume and the volume of liquid in the accumulator at filling temperature.

The LHEX is a plate fin cross-flow type heat exchanger where the model parameters length, width, and height determine the total heat transfer area of both the hot and cold side. Such a heat exchanger is a common and appropriate selection for gas-to-liquid applications, where the optimal arrangement conforms to maximizing the surface area on the gas side. The presented component model is founded on the theory presented by Kays et al. in (Kays and London 1984).

The assumed flow arrangement in this model component is that of one fully mixed fluid (the gas) and one unmixed fluid (the liquid). This assumption translates to that the gas temperature is constant perpendicular to the direction of the flow. The hot and cold side, $i = h$ and $i = c$ respectively, heat transfer rates can be expressed as

$$h_i = \left[\frac{C_p S_t \dot{m} \frac{4l}{D_h}}{A} \right]_i \tag{5}$$

where A_i is the total heat transfer area of side i. The Stanton number S_t in Equation 5, named after Thomas Stanton, is a heat transfer modulus that is used to characterize heat transfer (Ackroyd 2007). In this particular model, the Stanton number is seen as a tunable exponential function of the Reynolds number that is calibrated against efficiency measurements.

The LHEX overall thermal resistance can now be expressed as

$$R = \sum_{i=c}^{h} \frac{1}{h_i A_i}. \tag{6}$$

assuming negligible thermal resistance of the walls and no extended fin surface on either the hot or cold sides of the LHEX.

The thermal efficiency, for a heat exchanger with this particular assumed flow arrangement, is expressed as either

$$\varepsilon = 1 - e^{-\left[1 - e^{-N_{tu} \frac{C_{min}}{C_{max}}}\right] \frac{C_{max}}{C_{min}}} \tag{7}$$

or

$$\varepsilon = \frac{C_{max}}{C_{min}} \left[1 - e^{-\left[1 - e^{-N_{tu}}\right] \frac{C_{min}}{C_{max}}} \right] \tag{8}$$

depending on which of the side's flow capacity rates C that are limiting. If $C_{min} = C_c = \dot{m}_c C_{p_c}$ then Equation 7 is applicable, and if $C_{min} = C_h = \dot{m}_h C_{p_h}$ then Equation 8 is applicable. The thermal resistance influences the efficiency via the heat transfer parameter

$$N_{tu} = \frac{1}{R C_{min}} \tag{9}$$

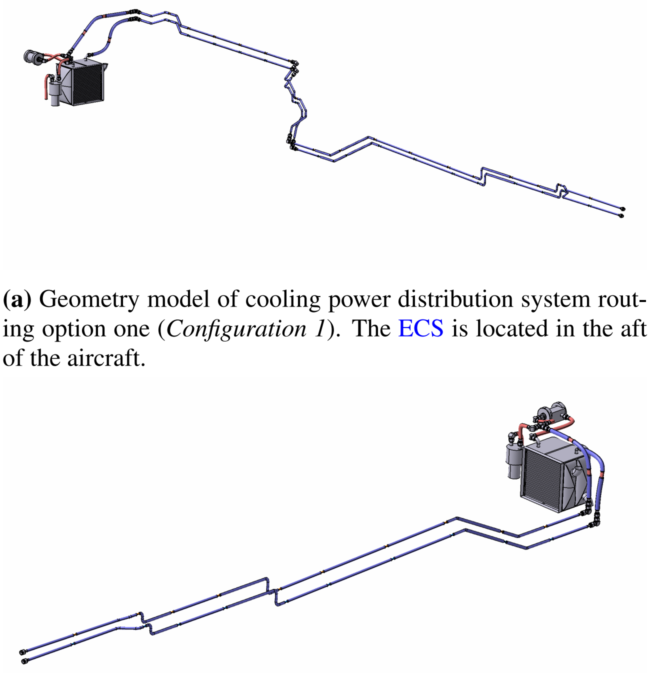
which connects the geometrical parameters to the efficiency. In the application example LHEX model $l_h = 2 \cdot b$, as the hot liquid passes the cold side surface twice, and $l_c = l$. Additionally, the parameters b, h, and l affect the efficiency through S_t which here is modeled as an exponential function of the Reynolds number. This exponential function is tuned such that the model complies with supplier data.

Finally, the LHEX hot side outlet temperature T_h^{out} can be described as function of the efficiency

$$T_h^{out} = T_h^{in} - \varepsilon (T_h^{in} - T_c^{in}) \tag{10}$$

where the superscipt indicates inlet or outlet temperature.

5.2 Geometrical representations



(a) Geometry model of cooling power distribution system routing option one (*Configuration 1*). The ECS is located in the aft of the aircraft.

(b) Geometry model of cooling power distribution system routing option two *Configuration 2*. The ECS is located immediately behind the aircraft's cockpit.

Figure 5. Use-case geometry models representing the two different routing options under investigation. The piping reaches from the LHEX to the front of the aircraft where the radar is located. The radar is not included in either Figure 5a or Figure 5b.

Two different configurations of the coolant distribution system are modeled in CATIA, see Figure 5. The resulting geometry models include geometrical representations of all the parts of the coolant distribution system model. The main components, the LHEX, Pump, and Accumulator are the same in both configurations.

In both configurations, the piping reaches from the LHEX to the front of the aircraft where the radar is located. The main difference between the configurations lie in the positioning of the ECS and coolant distribution system core, i.e. the accumulator, pump, and LHEX. In *Configuration 1*, the routing extends from the aft, via the aircraft ridge, to the radar. This configuration results in a significantly longer routing with more bends compared to *Configuration 2*, where the ECS is located immediately behind the cockpit. Both configurations have advantages and disadvantages. For example, the potential increased pressure drop of *Configuration 1* could be outweighed by the reduced need for transporting engine bleed air to the, in this case, bleed-air-driven ECS.

6 Use-case

A use-case, presented in this section, is formulated to demonstrate the functioning and benefits of the developed technology. An Operational Concept (OpsCon) (International Council on Systems Engineering 2015) mission, along with a sub system requirement posed by the hypothetical developer of the application example radar, together compose the use-case requirements on the coolant distribution system.

6.1 Prerequisites

The application example described in Section 5 naturally serves as the primary use-case prerequisite. Here, the application example is available in the form of a generic SSP, including a template SSM file generated using the functionality provided in Listing 2.

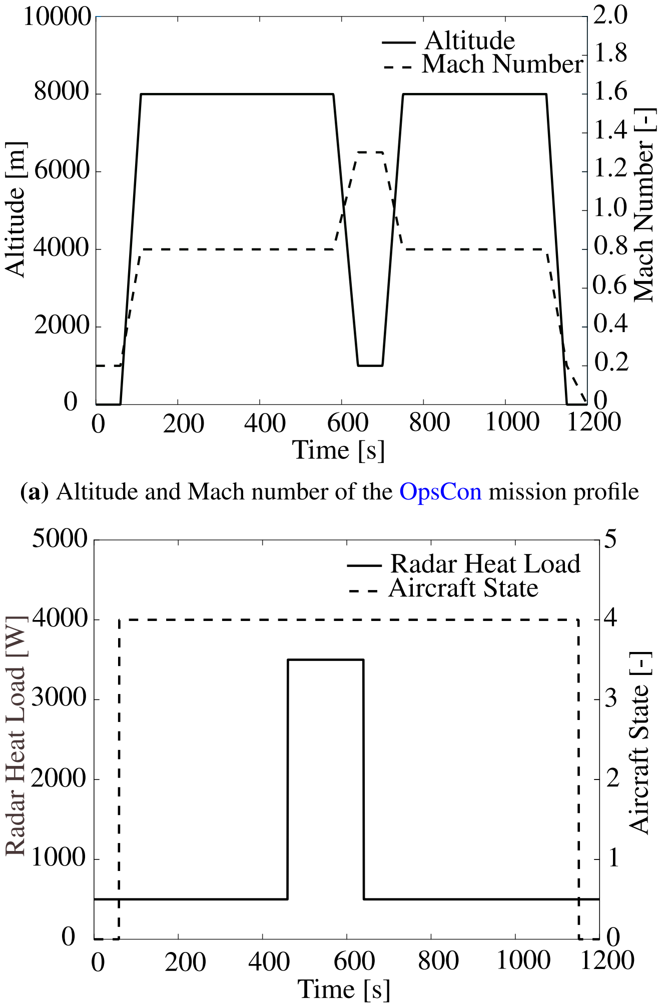
In addition, the OpsCon mission is seen as a top-level requirement which the application example should fulfill. The application example boundary conditions of the OpsCon mission is presented in Figure 6. The mission altitude and Mach number profiles are presented in Figure 6a, and the radar heat load and SW input aircraft state in Figure 6b.

The application example aircraft leaves the runway after approximately 100 s with the goal of identifying an unknown aircraft known to be present approximately 115 km from the base. A climb and acceleration to cruise conditions are then initiated and realized. The aircraft operates at cruise conditions until it reaches the specified location. A loitering phase is commenced and the radar is shifted from stand-by to active, see the power transient depicted in Figure 6b. The aircraft being sought is located after 60 s of searching and the operating conditions are then matched to those of the foreign aircraft via a speed increasing dive. Once contact has been established, the radar is shifted to stand by mode and the aircraft is returned to base via a second low fuel consumption cruise phase.

The OpsCon mission specifies the use and functioning of a radar. This radar functions provided that the subsystem requirement

- The difference in radar coolant inlet and outlet temperature shall not exceed 13°C

is fulfilled throughout the OpsCon mission.



(a) Altitude and Mach number of the OpsCon mission profile

(b) Heat load exerted by the application example radar component during the OpsCon mission. The radar can here operate in two different discrete modes: a stand by mode corresponding to 500 W of power and a active mode corresponding to 3500 W. The dashed line represents the SW input signal *AircraftState* which indicates whether the aircraft is situated on the ground (*AircraftState*= 1), or if it is airborne (*AircraftState*= 4).

Figure 6. Boundary conditions corresponding to the specified OpsCon mission profile

6.2 Sunny day scenario and expected outcome

The engineering team responsible for the acquisition and tailoring of the ECS to be used in the aircraft, in collaboration and agreement with the ECS supplier, has identified two different possible system locations: below the fin in the aft of the aircraft, and immediately behind the cockpit.

Each ECS position results in a different routing of the liquid coolant distribution system as the consumer of coolant power is located in the nose of the aircraft. The engineer responsible for specifying the installation of the liquid coolant distribution system is proposing two different routing options, see Figure 5 and Section 5.2. Geometry models are developed for both of the two different routing options and the corresponding parameters are exported, using the functionality presented in Section 3, as two different SSV files. These SSV files are placed in the resources of the application example SSP as shown in Figure 3b. An extract of the exported geometry information in the SSV format is provided in Listing 5. An extract of the corresponding intermediate CATIA XML is provided in Listing 6. The presented parameter value is common to both *Configuration 1* and *Configuration 2*.

Listing 5. Extract of application example geometry information in the SSV format.

```
<ssv:ParameterSet name="product_ECS">
  <Units>
    <Unit name="m">
      <BaseUnit m="1"/>
  </Units>
  <ssv:Parameter name="part_LHEX.
      parameterSet_inputParameters.
      parameter_width">
    <ssv:Real unit="[m]" value="0.3"/>
  </ssv:Parameter>
</ssv:ParameterSet>
```

Listing 6. Extract of application example geometry information in the intermediate XML format described in Section 3.1.

```
<product name="ECS">
  <part name="LHEX">
    <parameterSet name="inputParameters">
      <parameter name="width" type="Double">
        <value>300</value>
        <unit>[mm]</unit>
      </parameter>
    </parameterSet>
  </part>
</product>
```

In parallel, the involved stakeholders agree upon a mapping between parameter values and the parameters of the FMUs relevant to the application example; thus updating the SSM from template to the final version of the instantiated SSP.

The sub-system requirements need to be verified during the presented OpsCon. The analysis is suggested to provide feedback on the design in terms of suggestions concerning the ECS positioning and the accompanying routing. The feasibility is determined with respect to the presented system and sub-system level requirements.

7 Results and discussion

The application example is simulated for the mission profile described in Section 6.1. The geometry settings of the two different modeled configurations are summarized in Table 2 and Table 3. The parameters that differ between configurations are presented in Table 2. Parameter values that remain unchanged in the different configurations, but are specified by the geometry models, are presented in Table 3. The remaining system simulation model parameters are kept at their default values, as specified in their original M&S development environment.

	Feed line piping		Return line piping	
	l[m]	z[-]	l[m]	z[-]
Configuration 1	7.393	2.491	7.412	2.417
Configuration 2	4.614	0.985	4.571	0.880

Table 2. Summary of parameter values that differ between the two configurations. The cooling distribution system routing is subject to modification. The remaining coolant distribution system components, along with their constituent parameters, remain unchanged

	D_h[m]	h[m]	b[m]	l[m]	V_{acc}[m^3]
Piping	0.01				
Acc.					$1.71 \cdot 10^{-3}$
LHEX		0.3	0.3	0.3	

Table 3. Summary of parameter values that are specified by the geometry model but identical for the two different configurations.

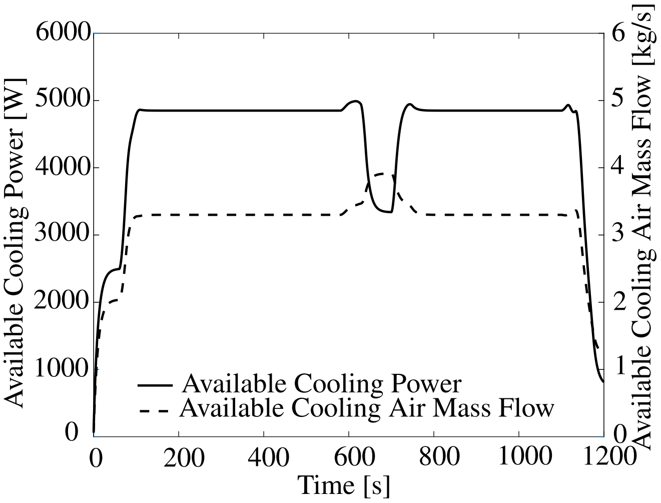
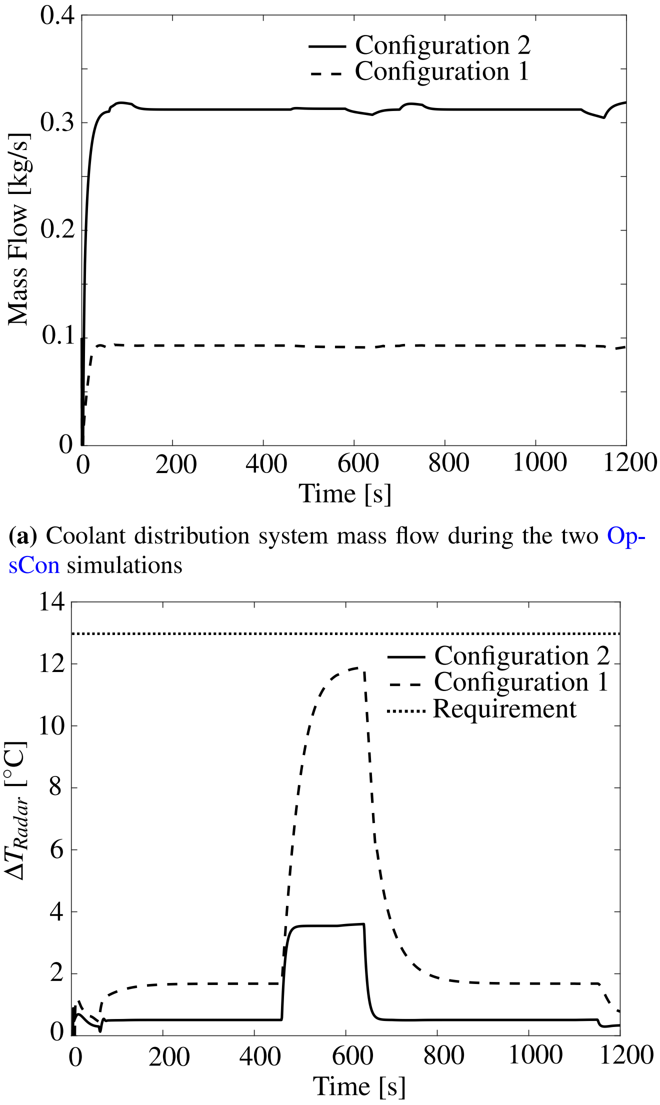


Figure 7. ECS model available cooling power (solid) and the available coolant mass flow as (dashed) for the OpsCon simulations

The simulation results used to assess the feasibility of the two different configurations, with respect to the requirements, are shown in Figure 7 and Figure 8. Figure 7 quantifies the performance limits of the included ECS model. The available cooling power is shown as solid in the figure and the available coolant mass flow as dashed. Note that the available cooling power is significantly greater than the OpsCon radar heat load, see Figure 6b, indicating that the ECS performance is sufficient for executing the mission.

The simulated radar inlet mass flow is shown in Figure 8a and the temperature increase over the radar in Figure 8b. The temperature increase remains below the required differential temperature level, dotted line in Figure 8b, throughout the simulated OpsCon for both configurations. Even so, the temperature increase is shown as significantly higher for *Configuration 1* than *Configuration 2*. This is a result of the corresponding lower levels of coolant mass flow, see Figure 8a. The coolant distribution mass flow depends on the pipe length l and pressure loss coefficient z, according to Equation 2, that are presented in Table 2.



(a) Coolant distribution system mass flow during the two OpsCon simulations

(b) Temperature increase over the radar during the two OpsCon simulations

Figure 8. Compilation of simulation result relevant during use-case requirement verification. The results stem from simulations of the two different configurations. *Configuration 1* is depicted as dashed, and *Configuration 2* as solid

8 Conclusions

There is much to gain already in adopting a single established standardized format for information exchange internally, within the confines of the organization, that can be version controlled and compared to previous versions using well established tools. This benefit can be increased if the standardized format is supported by the modeling tools such that the parameters can be automatically exchanged at manually, or automatically, generated events such as the commit of a model update to a repository.

The results of the research presented here indicate that the FMI and SSP standards show great promise for achieving such an automated simulation application development method. A method for exchanging parameter information between the engineering domains of system simulation and CAD has been established exploiting the, in this context, suitable open tools and standards. The method has been developed while keeping the aim of minimizing the impact on the modeling methodologies, mathematical or geometrical, in mind. The application example's aggregated pressure loss coefficients, for example, could have been computed in the components of the modeling library compared to in the developed VBA macros, see Section 3. This would, however, constitute a major change to any library that is mature and used in several different models.

The presented method has been contextualized to the simulation model development and maintenance processes currently deployed at Saab Aeronautics. Furthermore, the work has resulted in the specification of necessary functionality for manipulating SSPs. Prototype functionality is implemented in, and tested using, the OMSimulator tool. The presented methodology would benefit greatly if the presented functionality were made available in the modeling tools best suited for each considered modeling domain.

Additionally, the presented work targets the exchange and specification of parameters exposed at the interface of FMUs. An FMU generated from Modelica only allows modification of *non-structural* parameters, i.e. parameters that do not impact upon the internal structure of the system of equations. This delimitation could be avoided if the available M&S tools developed SSP support not only coupled to FMI but also to the tool's native modeling language. In such a case, the parameters could be exchanged prior to code generation and compilation.

Acknowledgements

The presented research was conducted within the frame of the ITEA3 project *EMBrACE* and the NFFP7 project *Digital Twin for Automated Model Validation and Flight Test Evaluation*. The research was funded by Vinnova and Saab Aeronautics. The authors would specifically like to thank Lennart Ochel and Arunkumar Palanisami for their help in implementing the necessary OMSimulator functionality. In addition, the authors would like to thank Dan Louthander for his help with the details of the application example and reviewing of the manuscript.

References

Ackroyd, J. A. D. (2007). "The Victoria University of Manchester's contributions to the development of aeronautics". In: *The Aeronautical Journal (1968)* 111.1122, pp. 473–493. DOI: 10.1017/S0001924000004735.

Andersson, Henric and Magnus Carlsson (2012). *Saab Aeronautics Handbook for Development of Simulation Models : Public Variant*. Tech. rep. 12/00159. Linköping University, Machine Design.

Auslander, D. M. (1968). "Distributed System Simulation With Bilateral Delay-Line Models". In: *Journal of Basic Engineering* 90.2. DOI: https://doi.org/10.1115/1.3605079.

Baumgartner, Daniel and Andreas Pfeiffer (2014-03). "Automated Modelica Package Generation of Parameterized Multibody Systems in CATIA". In: *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*. Linköping University Electronic Press. DOI: 10.3384/ecp14096913.

Eek, Magnus, Hampus Gavel, and Johan Ölvander (2017-02). "Definition and Implementation of a Method for Uncertainty Aggregation in Component-Based System Simulation Models". In: *Journal of Verification, Validation and Uncertainty Quantification* 2.1. DOI: https://doi.org/10.1115/1.4035716.

Elmqvist, Hilding, Sven Erik Mattsson, and Christophe Chapuis (2009-10). "Redundancies in Multibody Systems and Automatic Coupling of CATIA and Modelica". In: *Proceedings of the 7 International Modelica Conference Como, Italy*. Linköping University Electronic Press. DOI: 10.3384/ecp09430113.

Engelson, V., H. Larsson, and P. Fritzson (1999). "A design, simulation and visualization environment for object-oriented mechanical and multi-domain models in Modelica". In: *1999 IEEE International Conference on Information Visualization (Cat. No. PR00210)*. IEEE Comput. Soc. DOI: 10.1109/iv.1999.781557.

FMI Development Group (2020-12-15). *Functional Mock-up Interface for Model Exchange and Co-Simulation*. Report 2.0.2.

Fritzson, Peter (2004-01). *Principles of Object Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press. ISBN: 9780470545669. DOI: 10.1109/9780470545669.

Fritzson, Peter et al. (2020). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control: A Norwegian Research Bulletin* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Hällqvist, Robert (2019). "On Standardized Model Integration : Automated Validation in Aircraft System Simulation". Licentiate Thesis. Linköping University, Faculty of Science and Engineering. ISBN: 9789179299293. DOI: 10.3384/lic.diva-162810.

Hällqvist, Robert et al. (2018). "A Novel FMI and TLM-based Desktop Simulator for Detailed Studies of Thermal Pilot Comfort". In: *Proceedings of the 31st Congress of the International Council of the Aeronautical Sciences*. International Council of the Aeronautical Sciences. ISBN: 978-3-932182-88-4.

International Council on Systems Engineering (2015). *Systems Engineering Handbook*. 4th ed. John Wiley and Sons, Inc. ISBN: 9781118999400.

Kays, W.M. and A.L. London (1984). *Compact heat exchangers*. Krieger. ISBN: 9781575240602. URL: http://books.google.de/books?id=A08qAQAAMAAJ.

Krus, Petter et al. (1990-01). "Distributed Simulation of Hydromechanical Systems". In: *Third Bath International Fluid Power Workshop*.

Lind, Ingela and Alexandra Oprea (2012-11). "Detailed geometrical information of aircraft fuel tanks incorporated into fuel system simulation models". In: *Proceedings of the 9th International MODELICA Conference, September 3-5, 2012, Munich, Germany*. Linköping University Electronic Press. DOI: 10.3384/ecp12076333.

Ljung, Lennart and Torkel Glad (2004). *Modelbygge och Simulering*. Vol. 2. Studentliteratur. ISBN: 91-44-02443-6.

Miller, Donald (1990). *Internal Flow Systems*. Cranfield, Bedford: BHRA (Information Services. ISBN: 978-0956200204.

Modelica Association (2019-03-05). *System Structure and Parameterization*. Report 1.0.

Modelica Association Project System Structure and Parameterization (2021). *System Structure and Parameterization*. URL: https://ssp-standard.org (visited on 2021-05-08).

Munjulury, Raghu Chaitanya (2017). "Knowledge-Based Integrated Aircraft Design : An Applied Approach from Design to Concept Demonstration". Linköping. ISBN: 9789176855201.

Munjulury, Raghu Chaitanya et al. (2016). "A knowledge-based integrated aircraft conceptual design framework". In: *CEAS Aeronautical Journal* 7.1, pp. 95–105.

Ochel, Lennart (2021). *OMSimulator's documentation*. Accessed: 2021-02-18. URL: https://openmodelica.org/doc/OMSimulator/master/html (visited on 2021-02-18).

Ochel, Lennart et al. (2019-03-04). "OMSimulator – Integrated FMI and TLM-based Co-simulation with Composite Model Editing and SSP". In: *Proceeding of the 13th International Modelica Conference*. DOI: 10.3384/ecp1915769.

OpenCPS Project Partners (2019). *Project 14018:Open Cyber-Physical System Model-Driven Certified Development*. Accessed: 2018-06-21. URL: https://www.opencps.eu/ (visited on 2019-11-15).

Remond, Xavier, Thierry Gengler, and Christophe Chapuis (2015-09). "Simulation of Piping 3D Designs Powered by Modelica". In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. Linköping University Electronic Press. DOI: 10.3384/ecp15118517.

Roy, Christopher J. and William L. Oberkampf (2011-06). "A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing". In: *Computer methods in applied mechanics and engineering* 200.25-28, pp. 2131–2144. DOI: 10.1016/j.cma.2011.03.016.

Roza, Manfred, Jeroen Voogd, and Derek Sebalj (2012-10). "The Generic Methodology for Verification and Validation to support acceptance of models, simulations and data". In: *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology* 10.4, pp. 347–365. DOI: 10.1177/1548512912459688.

Stokes, Melody (2001). *Managing engineering knowledge : MOKA: methodology for knowledge based engineering applications*. Professional Engineering Publ. ISBN: 1860582958.

The Modelica Association (2019). *Modelica and the Modelica Association*. Accessed: 2018-06-21. URL: https://www.modelica.org/ (visited on 2019-11-15).

Modelica, FMI and SSP for LOTAR of Analytical mBSE models: First Implementation and Feedback

Clément Coïc[1] Adrian Murton[2] Juan Carlos Mendo[3] Mark Williams[3] Hubertus Tummescheit[1] Kurt Woodham[4]

[1]Modelon, Sweden, {clement.coic, hubertus.tummescheit}@modelon.com

[2]Airbus Operations Ltd. United Kingdom, adrian.murton@airbus.com

[3]The Boeing Company, USA, {juan.c.mendo, mark.williams}@boeing.com

[4]NASA Langley Research Center, USA, kurt.woodham@nasa.gov

Abstract

LOng Time Archiving and Retrieval (LOTAR) of models is key to using the full capabilities of model-Based System Engineering (mBSE) in a system lifecycle – including certification. The LOTAR MBSE workgroup is writing the EN/NAS 9300-Part 520 to standardize the associated process, in the aeronautics industry, and suggests the usage of Modelica, FMI and SSP standards for its purpose. Acceptance of such a process requires a match between industrial needs and software vendor implementations. This is helped by a tool-agnostic implementation of the process and following specific adaptations within the Modelon Impact software. This initiative – inside the LOTAR workgroups – highlights the suitability of such a process but also points at flaws or overhead due to the lack of connection between the Modelica, FMI and SSP standards, as well as the MoSSEC (ISO 10303-243) standard. The recommendations proposed in this document could have a significant impact on the final adoption of the LOTAR standard – relying on Modelica, FMI and SSP standards.

Keywords: Archiving, Retrieval, LOTAR, mBSE, MoSSEC, FMI, SSP

1 Introduction

Contrary to the software industry where end-of-life is programmed and conversion to a newer alternative is “enforced,” the industrial products containing complex cyber-physical systems have longer lifecycles and associated maintenance.

In the aerospace industry, designing an aircraft takes about a decade. The production cycle averages three decades, and their service life extends for another three decades. Deciding to develop a new aircraft is a choice that impacts two thirds of the next century. The technology and design choices, the system and component sizing, the rationale and arguments in each decision taken shall be stored and kept accessible during the aircraft’s entire lifecycle. This enables its potential evolutions and design reuse opportunities. It capitalizes on the work and knowledge and supports a response to future questions. Other key aspects of data archiving and retrieval in the aerospace industry is to provide a basis for the certification of future modifications, address component obsolescence, and support accident investigations.

These are the challenges that the LOTAR international consortium of Aerospace manufacturers –jointly facilitated by AIA, ASD-Stan, AFNeT, prostep ivip and PDES, Inc. – are facing through the creation and deployment of the EN/NAS 9300 series of standards for long-term archiving and retrieval of digital data. To ensure industry adoption, the resulting process must be based on standardized practices and proven solutions, listed on the website (LOTAR International 2021). The authors of this papers are active members of the “MBSE workgroup” within the LOTAR consortium.

Model-Based System Engineering (MBSE) definition, *"the formalized application of modeling to support system requirements, design, analysis, verification and validation beginning in the conceptual design phase and continuing throughout the development lifecycle"* (INCOSE Operations 2007) is widely used in the aerospace industry.

While MBSE includes all types of models, the authors previously introduced the acronym “mBSE” [or “little m BSE” as opposed to the “big M BSE”] to narrow the focus on the preservation of system-descriptive and analytical models that are explicit, coherent, and consistent (Nallon 2021). The integrated models provide high-fidelity, rich representations – potentially of different granularity of sub-systems. These models are viewpoints that support the decisions affecting the product’s architecture, new technologies, or component sizing. This distinction is necessary to separate this work from other types of models, e.g. 3D CAD models. (This does not mean that these models should be decoupled.)

The archiving and retrieval of the models developed is mandatory to utilize the efforts and rationales the model served throughout the lifecycle of the (cyber-physical) system it represents. mBSE data is also applicable to the certification process and in-service maintenance of Aerospace products. The need for long-term archiving is an existing regulatory requirement. As many design representations shift to a digital format the urgency of defining archiving standards is in response to a critical industry need. EN/NAS 9300 Part 520 standardizes the long-term archiving and retrieval process for analytical mBSE models.

Section 2 of this paper concisely introduces the suggested steps for archiving and retrieval and discusses the supplemental needs of the data archive and model manifest. Section 3 presents a tool-agnostic implementation and its integration within Modelon Impact. Section 4 discusses the prototype results and proposes specific recommendations for the standards being used.

2 Archiving and retrieval process

2.1 Abstract and Keywords

While the EN/NAS 9300 Part 520 describes the archiving and retrieval process in more detail, the main points are listed below.

Archiving

- Develop and validate an mBSE model,
- Create an associated meta-data manifest,
- Export the model as an FMU or SSP,
- Include the manifest in the “extra” folder of the FMU or SSP
- Archive the FMU or SSP, together with its manifest, in the archiving platform/repository,
- Populate the AIP (Archive Information Package) with information from the manifest

Retrieval

- Access the AIPs on the archiving platform
- Select the desired archived FMU/SSP by examining the repository’s AIPs
- Retrieve the FMU/SSP
- Consult the associated manifest to validate the retrieval results
- Verify that the model is not corrupt

For this standard to be easily deployable, the emphasis is on the archiving and retrieving process of the model, not its creation. A tool vendor is welcomed and even encouraged to implement some of these steps earlier in the modeling process. In a typical scenario, the model manifest is populated early in the model’s lifecycle. The population process is typically iterative throughout the product design phase and could be optimized to support additional goals such as model exchange. These steps will typically happen prior to the model’s export as an FMU for archival purposes. However, as long as the consistency of the model and the meta-data is sustained, the order of operations is not imposed.

2.2 Relying on existing standards

The LOTAR MBSE workgroup made an extensive effort to reference and map most related standards, their applicability, usage, and maintenance (Williams 2021). One aim was to define which standards to rely on for the archive and manifest formats. The first consideration was a neutral format with the widest potential tool support for long term archiving. The second consideration was endorsement by the Aerospace OEMs. It was found that:

- The FMI standard has reached a level of maturity and availability that supports model archiving and preservation.
- The SSP standard – being the structured system variant of the FMI standard – is also a recommended alternative for system model archiving and preservation.
- The ISO STEP AP243 (MoSSEC 2021) standard, in a format similar to the Model Identity Card (MIC), is the recommended format for the model manifest.

These mature tool-agnostic standards form a solid base for the Part 520 standard on which any tool vendor can build a marketable solution.

2.3 LOTAR manifest

The LOTAR manifest is the identity card that enables each model to travel and be identified uniquely. In the first tool-agnostic implementation presented by this paper the manifest is stored as an XML (W3C 2006) file with tags arranged in various categories, which gather attributes and associated values.

As represented in Listing 1., the simplicity of the XML mark-up language makes it a suitable candidate for parsing of the manifest or performing further action on it – e.g. generation of a graphical representation, or populating repository attributes.

Listing 1. A short extract from a sample manifest:

```
<LOTAR_Manifest>
  <GeneralPLM>
...
  </GeneralPLM>
  <DevelopmentIntegrationAndExecution>
...
  </DevelopmentIntegrationAndExecution>
  <PhysicsContentAndUsage>
    <PhysicsContentProperties
      Dimension=""
      PhysicsDomain=""
      Timescale=""
      Linearity=""
      ModelType_Usage="">
    </PhysicsContentAndUsage>
    <ValidityRange
      ValidityRange="">
    </ValidityRange>
    <ModelFidelity
      RepresentedPhenomena=""
      NeglectedPhenomena="">
    </ModelFidelity>
  </PhysicsContentAndUsage>
...
  <ModelVariables
  </ModelVariables>
  <VerificationAndValidation
...
  </VerificationAndValidation>
</LOTAR_Manifest>
```

The different categories in the manifest aim to capture the design intent of the model (what), the rationale and purpose for creating the model (why), the content, fidelity, and format of the model (what/how), as well as its provenance (who/when).

These specific metadata categories can be mapped to ISO STEP AP243 (MoSSEC), and they are used to capture the systems engineering context around each model to be archived. The MoSSEC specification standardizes such context. Sharing or archiving, the MoSSEC-styled metadata information facilitates every model's availability, retrieval, and reusability

3 First implementation

The LOTAR mBSE workgroup solicited Modelon to implement a tool-agnostic version of the archive and retrieval process. The aim was to detect the "paper cuts" in the process that could hinder the standard's adoption. Performing this type of early prototyping, prior to the standard's initial release, verifies the viability of the identified mBSE data standards and the associated workflow needed for archiving, sharing, and retrieving analytical models.

3.1 Definition of the prototype system

As this work focuses on the archiving and retrieval process, the prototype system is made as simple as possible while still illustrating the optional process steps. The LOTAR mBSE workgroup selected the Regulated Actuator system represented in **Figure 1** for their proof of concept.

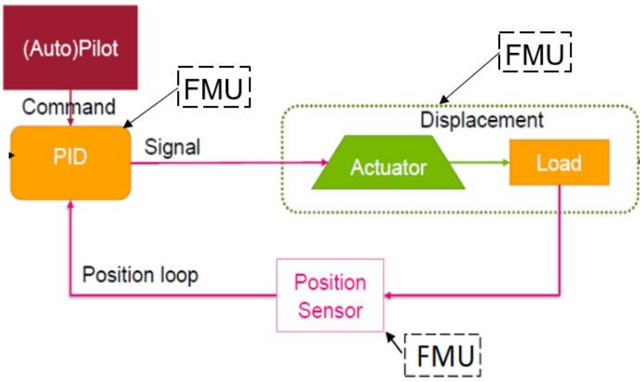


Figure 1. Regulated Actuator System

The diagram represents a system whose goal is to control the position of a flight control surface to simplify the (auto) pilot's response when facing external loads. The system is composed of three subsystems:

- The controller, a simple PID (proportional, integral, derivative) controller that receives the (auto) pilot position commands and the measured actuator position. It then outputs a command based on the errors detected between both.
- The plant model gathers both the actuator and load models. An output is the actual surface position.
- A sensor model inserts a delay in the measurement.

The three models are highly simplified representations of the physics involved. The focus is to define a common process independent of the tools and potentially different environments that could be used by different teams. Furthermore, each subsystem is modeled separately and exported as an FMU; the overall system can be exported as a SSP – by composition of the previously built FMUs.

3.2 A tool-agnostic implementation

3.2.1 Modelica models and FMU generation

The three models described in section 3.1 and presented in **Figure 1** are developed using the Modelica language. Once more, the reason is not to take advantage of the language capabilities but to take advantage of its openness. Indeed, the models are deliverables to the prototyping effort and can be shared in text format for further use. A derived advantage is the inclusion of the documentation-annotation provisions in MLSv35r0 (Modelica Association 2021) that enables embedding the model description and experimental frame, within the model itself, in HTML format.

The FMUs (Functional Mockups Units) are generated using the Modelon compiler included in Modelon Impact but could have been generated by any other Modelica compiler, e.g., OpenModelica. They have been generated in both the Model Exchange and Co-Simulation formats for further study and use.

Note: for LOTAR purposes, a co-simulation FMU has a clear advantage because the solver is stored within the FMU – by definition. This is a highly relevant point for long term archiving. Which solvers will be available in 50 years from now? How will one be able to couple them with another FMU? Nevertheless, the model exchange FMU also has obvious advantages because they are great candidates for a direct coupling into an SSP prior to archiving. The results of this prototype will be analyzed by the LOTAR team to understand the preferred archive alternatives, and the best potential formats for archiving. A Modelica language tool may not always provide the source, and entire repositories of archived formats may need to be converted to alternative standards in the future.

The following steps of the process are achieved by relying on Python scripts – using open-source or, at least, free of use packages. As a reminder, this is an implementation that helps to bring the standard to life but not the standard itself – anyone is free to implement it using different means.

3.2.2 Creation of the LOTAR manifest

The LOTAR manifest being written is an XML file (W3C 2021), created using the Python module xml.etree.ElementTree. The XML tree structure is built first and then attributes are set to their values. Unknown values are left as empty strings.

In the future envisioned process, populating the manifest metadata should be a semi-automatic process using a tool specific implementation that occurs prior to the start of the archival process.

However, the EN/NAS 9300 Part 520 standard has not been formally released yet, so this is currently performed manually. Nevertheless, this first implementation highlighted many commonalities with the “modelDescription,xml” file contained in the FMUs. The PyFMI (PyFMI 2020) package is used to load and interact with the FMU in its most basic form: accessing the modelDescription information. This way, many fields of the manifest are automatically populated by the Python script. A simple extract is defined in Listing 2.

Listing 2. A short extract from a sample manifest:

```
# Root
manifest=ET.Element('LOTAR_manifest')
# LOTAR_manifest > GeneralPLM
GeneralPLM=ET.SubElement(manifest,'GeneralPLM')
# LOTAR_manifest > GeneralPLM > ...
ProvenanceOwnershipDate=ET.SubElement(GeneralPLM,
'ProvenanceOwnershipDate')
# Populate creation date
ProvenanceOwnershipDate.set('Created_on',
model.get_generation_date_and_time())
```

Note that another option would be to extract and parse the XML file directly. This would be, however, more laborious and the solution presented relies on maintained python packages, and thus is more convenient.

At this point, it becomes important to note that a subset of model manifest fields can be taken straight from the “modelDescription.xml”. However, the model manifest offers extra metadata that travels with the original model, regardless of its format: FMU or native.

3.2.3 Inclusion of the manifest in a Functional Mockup Unit

Because the FMU is a zip file, the inclusion of the manifest can be performed automatically with a Python module such as zipfile (Pyhton Software Foundation 2021). Three specific points are listed here:

- To prevent conflicts, the naming of the manifest uses reverse domain notation such as `extra/org.mossec/ LOTAR_Manifest.xml`
- Ensure the manifest is added in the “extra” folder of the FMU recently available FMIv2.0.2 (Modelica Association, 2020) in the standard.
- To minimize the file corruption risks, it is recommended to open the FMU in a mode in which it is only possible to append new files, not to modify existing ones.

The FMU is now ready for archiving on the platform.

3.2.4 Accessing the manifest

Once the FMU is archived, the manifest should be accessible by the repository, so the contained information enables the user to identify the correct model needed for retrieval. The selected implementation consists of inverting the previous process: open the zip in read only mode, extract a copy of the manifest in a preselected location in the repository and then access this copy. This can be achieved with the same Python module zipfile.ZipFile.

Note that one recommendation for the repository could be to perform this manifest manipulation when archiving the FMU, keeping a pointer toward the original source file – thus collecting an organized set of manifests. Retrieval could then consist of browsing the set of stored manifests and selecting the correct one. The archiving platform could then access the related FMU, verify that the same manifest is included and perform the retrieval. From a LOTAR perspective, the manifest is very similar to the AIP mentioned previously and could be replicated accordingly. This would ensure the model metadata would exist both internally and externally to the FMU zipfile.

3.2.5 Repeating the process at system level

The prototype was designed so that a system level example could be studied and refined. The regulated actuator model is built as an SSP by composition of the existing FMUs. The manifest template, as defined in the first version of the EN/NAS 9300-Part 520, applies primarily to subsystem level component models and is not directly applicable yet to system level simulations or setups. The Python scripts were adapted to reach the manifests inside the FMU zip files when archiving. In the future, the manifest of a structured system model should be defined and stored in the “extra” folder of the SSP – for consistency.

3.2.6 Availability of developed models and code

The LOTAR MBSE workgroup and Modelon agreed to make these models and Python code available to the LOTAR and eventually to the broader community. This is an added benefit of this first implementation: publicly available basic models and Python scripts – relying on open-source or free of use packages – that follow and implement the process from the draft Part 520 standard. This way, there are no proprietary restrictions preventing a tool vendor from implementing the standard. Modelon illustrates this fact with the addition of a custom function within Modelon Impact to write and add the manifest when exporting an FMU.

3.3 A tool-specific implementation

3.3.1 Modelon Impact in four sentences

Modelon Impact is a cloud native based modeling and simulation environment relying on and enhancing open technologies such as Modelica, FMI or Python (Modelon 2020). Modelon Impact is aimed at democratizing simulation to a broader audience by providing a user friendly, yet powerful interface to develop and simulate models or utilize them in a very narrow context, including the use of web applications (Coïc 2020a). Modelon Impact offers both steady-state and dynamic simulation capabilities (Coïc 2020b), which exposes more opportunities to the model developer to make a model best suited to the model user. Finally, it is possible to implement user specific workflows in Modelon Impact through Python-based custom functions which can interact with the Modelon Impact API.

3.3.2 A custom function for the manifest generation

One convenient way to implement a Modelon Impact specific implementation of this process is to develop a dedicated custom function that would write and add the manifest to the FMU when compiling a model.

The tool-agnostic prototype is Python-based and relying uniquely on open-source technologies. The step to implement a Modelon Impact custom function is thus minor as these also rely on the Python language.

Custom functions are available in Modelon Impact by hovering over the simulate button (see **Figure 1**) or by selecting the dedicated function in the experiment mode (see **Figure 2**). In the latter case, more actions are possible through user inputs. For example, it was decided to add the “Author name” and “Organization” as a user input fields so that this information would always be added to the manifest.

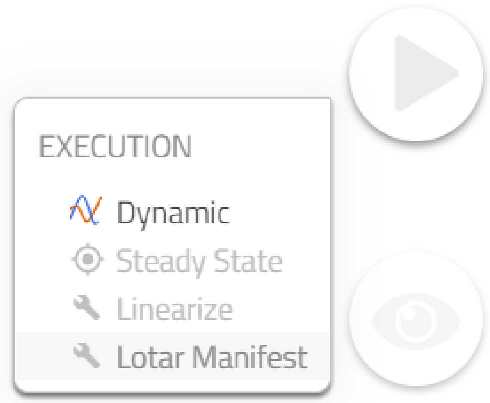
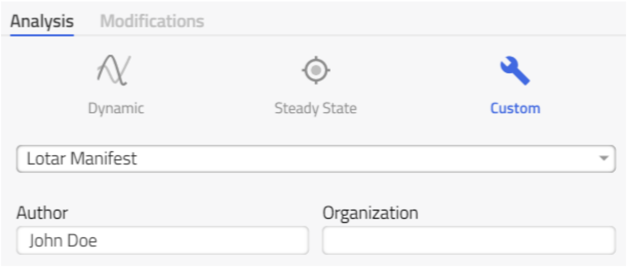


Figure 1. Direct generation of LOTAR manifest in Modelon Impact



Lotar Manifest

Author: John Doe

Organization:

Figure 2. Generation of the LOTAR manifest after optional user input provision, in Modelon Impact

Notes:

- This is hard to demonstrate in a paper, but a short demonstration of the python interface using PyFMI (PyFMI 2020) is available.
- The rest of the process is independent of the modeling and simulation environment but occurs on the archiving platform. Therefore, Modelon limits their tool-specific implementation to this step.

4 Discussions and recommendations

The completion of this prototype, prior to the release of the Part 520 process standard, brought great insights on the applicability of the representative workflow and the potential need for future improvements. The following recommendations and criticisms concern both the Part 520 and the Modelica, FMI and SSP data standards. This paper is directed toward the users of these technologies.

4.1 Self-criticism

Constructive criticism is what ensures quality of work. This prototype was developed with this in mind: stress-test the process in order to spot any inconvenient or non-finalized steps. Listed below is a list of items that were identified as sources of improvement. The items are tagged with “minor” and “major” keywords to highlight their criticality. However, a “minor” item is not irrelevant as in the long run it could be a paper cut that prevents the standards future adoption.

- [Major] Create a proper mapping between the FMI "modelDescription.xml" and the LOTAR manifest. When filling out the manifest, many items were extracted from the PyFMI API. While this is convenient, this also shows some redundancies. A proper mapping would highlight whether the manifest should become a small extension of the existing modelDescription or remain a separate file.
- [Major] Define the content of the manifest for an SSP. Many of the fields identified for the FMU LOTAR manifest are less relevant at system level – especially if each FMU contains its own manifest. For example, what is the value of identifying the physical domains involved in an SSP if each FMU specifies its own? A substantial effort is expended by the industrials, under the guidance of prostep ivip to transform the "Glue Particle" into a data standard. This should expand the investigation of adding descriptive metadata to an SSP.
- [Major] How to precisely specify the format of the validation and verification scenarios. The LOTAR manifest includes these attribute fields but does not constrain their format. Specifying the file type for reference results would be a key to future success – especially when the validation tests are performed several decades later after retrieval from an archive.
- [Minor] Update the "Required" fields of the manifest. The manifest identified some fields as required (i.e. mandatory information) but the current version is not easily adapted to the FMI standard. For example, a field for "TargetTool_Name" is required while one benefit of the FMI standard is to be tool independent. Nevertheless, this field is relevant if some dedicated tests were performed in the future targeting a specific tool. This information could be maintained as a reference (although not "required").
- [Minor] Harmonize hierarchy and naming convention. Both "snake_case" and "camelCase" are used in the current LOTAR manifest sample. Some attributes repeat words in their names. This redundancy could be avoided by employing an additional hierarchical layer. This change is necessary to make the manifest more "attractive" for users and to remove sources of errors by using a formal naming convention.
- [Minor] Investigate how to add additional metadata to the manifest. It is expected that companies will need to define their own specific metadata that they will need to store within the manifest. This could be achieved in several different ways – e.g. by adding new attributes to existing tags or by defining new dedicated tags (for example, an "extra" tag?). This recommendation would be easy to implement.

4.2 Recommendations on used standards

The LOTAR MBSE workgroup remains confident that FMI and SSP standards provide a solid basis for the Part 520 standard. This makes the following constructive criticism even more relevant, as any improvement on these standards and/or associated tool implementations would also benefit the Part 520 indirectly. The criticality tags are also used here.

- [Major] Provide user entries for relevant metadata fields. When compiling an FMU or SSP, many of the "modelDescription" or "SystemStructure" fields are not defined, and the user is not provided with the choice to specify them. A simple example is the author field – that can be reached using PyFMI by "model.get_author()", where "model" represents the loaded FMU. For LOTAR purposes, the author's name is highly relevant, so are many other missing fields. It would be beneficial for the tool vendors to provide support for the missing fields.
- [Major] Improve support of SSP. While the FMI standard is highly supported by many tool vendors, the SSP standard lacks application support – only a few tools include the option. Several use cases (Thomas 2015) would benefit from wider SSP deployment. Long time archiving and retrieval of structured system models would offer additional options for the archivist.
- [Minor] Add a documentation bridge. Modelica is seen by the LOTAR MBSE workgroup as one of the main languages for model development. The Modelica specification includes a documentation-annotation that enables embedding the model description and experimental frame. Nevertheless, the effort the model developer expends when creating a Modelica model is lost when exporting the model as an FMU or SSP. The MoSSEC or MIC information would be extremely valuable. A path to explore would be the recommended approach to "compile" the documentation as HTML – similar to how it is done in libraries – in the resource folder of the FMU or SSP. Exporting only the top-level documentation would be appreciated by the model consumers.
- [Minor] Contribute to the reference results format specification. As discussed in the self-criticism section, it is relevant to specify how an FMU or SSP should be tested to validate its behavior and verify its integrity. This seems as relevant for the LOTAR purposes as it should be for the FMI and SSP standards.

- [Suggestion] Better support of metadata in the Modelica language. Allow the specification of many of the metadata (e.g. LOTAR fields) in a structured form in the Modelica model itself, maybe by embedding such a manifest. Then the manifest can be moved automatically from the source, to the FMU, and parts extracted to the SSP if needed. This would prevent changing the FMU after its generation, to add the manifest in the extra folder.

This section should act as a trigger for discussion or call for cooperation on these topics. The LOTAR MBSE workgroup would welcome any further joint actions with the Modelica Association and its members.

4.3 Further discussion

Several additional points are currently under discussion in the workgroup. Two are discussed here:

- In which form the model shall be archived? In the first implementation of the archiving process, Modelon included the Modelica source code as a resource in the FMU. This brings advantages for a future use of the model after retrieval. Nevertheless, what format of the model shall be stored in the archive is yet to be defined as this shall be generic to any software and prevent model corruption in the future.
- How can we ensure the framework to simulate the model will be available in the future? There are many dependencies for a model simulation: a compatible operating system, python packages, etc. Current discussions involve, for example, a "dynamic" archiving platform – that could perform regression tests of the stored at each dependency update – or to store an image/container of the dependencies together with the model. There seems to be a trade-of between heavy platform implementation and heavy archive files.

Validation and Verification of the LOTAR is another highly relevant point, which could have its own paper.

5 Summary and conclusions

The LOTAR MBSE workgroup aims at standardizing the long-term usage of models – driven by the aerospace industry's needs. The archiving process would also be applicable and valuable to other industries. A proper archiving and retrieval process would ensure model capitalization and reusability. Modelica is seen as one of the main languages for future model development, and the FMI and SSP standards provide a solid foundation for the EN/NAS 9300 Part 520 standard.

A prototype implementation of the process described in the Part 520 was conducted in both a tool agnostic way and within Modelon Impact – as a tool-vendor proof of concept. This work proved the suitability of the process and confirmed the LOTAR MBSE workgroup's recommendation of relying on FMI and SSP standards. This work also enabled identifying the next lines of actions on both the development of the Part 520 and the standards used – especially FMI and SSP. Recommendations addressed in this paper are from the perspective of any general user who may need to replicate this work. The LOTAR MBSE workgroup would welcome any further joint actions on the identified items.

References

Coïc C., Andreasson J., Pitchaikani A., Åkesson J. and Sattenapalli H., (2020). "Collaborative Development and Simulation of an Aircraft Hydraulic Actuator Model". Presentation: *Asian Modelica Conference*, Tokyo, Japan.

Coïc C., Hübel M. and Thorade M., (2020). "Enhanced Steady-State in Modelon Jet Propulsion Library, an Enabler for Industrial Design Workflows". Proceedings of the American Modelica Conference 2020, Boulder, Colorado, USA, March 23-25, 2020.

INCOSE Technical Operations, (2007). *Systems Engineering Vision 2020, Version 2.03.* International Council on Systems Engineering, San Diego, CA, USA. Technical Publication: INCOSE-TP-2004-004-02.

LOTAR International, (2021). "LOTAR Standard, Overview on Parts". URL: https://lotar-international.org/lotar-standard/

Modelon, 2020. Modelon Impact "*Lowering barriers and bridging gaps*". URL: https://www.modelon.com/modelon-impact-introduction/

Modelica Association, (2021). *Modelica – A Unified ObjectOriented Language for Systems Modeling. Language Specification Version 3.5, Revision 1*. Tech. rep. Linköping: Modelica Association. URL: https://www.modelica.org/documents/MLS.pdf [MOD21]

Modelica Association, (2020). *Functional Mock-up Interface, Standard specification*, Version 2.02. URL: https://github.com/modelica/fmi-standard/releases/download/v2.0.2/FMI-Specification-2.0.2.pdf

MoSSEC, (2021). "*Modelling and Simulation information in a collaborative Systems Engineering Context, Developer's Overview*". Data Standard: ISO 10303-243. URL: http://www.mossec.org/

PyFMI, (2020). "*Python package for loading and interacting with Functional Mock-Up Units*", Branch Version 2.6.x. URL: https://github.com/modelon-community/PyFMI

Thomas, E., Thomas, O., Bianconi, R., Crespo, M. and Daumas J., (2015). "*Towards Enhanced Process and Tools for Aircraft Systems Assessments during very Early Design Phase*". Proceedings of the 11th International Modelica Conference, Versailles, France.

Nallon J. & Williams M., (2020). "MBSE Tools Database Update, and Integrate Models with Tools". Presentation: INCOSE International Workshop, TIMLM Working Group, Torrance, CA, USA. URL: https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:incose_mbse_iw_2020:iw2020_timlm_mbseworkshop.pdf

W3C, (2006). "Extensible Markup Language (XML)", Version

1.1 (Second Edition), World Wide Web Data Standard. URL: https://www.w3.org/TR/2006/REC-xml11-20060816/

Williams M., Mendo J. and Nallon J., (2021). "*Where is your Roadmap for implementing MBSE Data Standards?*" Presentation: INCOSE International Workshop, TIMLM Working Group, Torrance, CA, USA. URL: https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:incose_mbse_iw_2021:iw2021_mbse-standards_timlm.pdf

Python Software Foundation, (2021). "*Working with zip archives*". The Python Standard Library, Data Compression and Archiving, documentation library. URL: https://docs.python.org/3/library/zipfile.html

eFMI: An open standard for physical models in embedded software

Oliver Lenord[1] Martin Otter[2] Christoff Bürger[3] Michael Hussmann[4]
Pierre Le Bihan[5] Jörg Niere[4] Andreas Pfeiffer[2] Robert Reicherdt[6] Kai Werther[7]

[1]Robert Bosch GmbH, Germany, [2]DLR-SR, Germany,
[3]Dassault Systèmes AB, Sweden, [4]dSPACE GmbH, Germany, [5]Dassault Systèmes SE, France,
[6]PikeTec GmbH, Germany, [7]ETAS GmbH, Germany

Abstract

This paper summarizes the final research results of the ITEA3 project EMPHYSIS (***em***bedded systems with ***phys***ical models ***i***n the production code ***s***oftware). Its core achievement is the new open *eFMI Standard* enabling automated workflows from high-level mathematical models of physical systems (referred to as physical models) to automotive compliant embedded software. eFMI (FMI for embedded systems) defines a container architecture for model exchange and testing. Multiple representations from an intermediate representation of sampled algorithms (GALEC) to production and binary code for specific embedded targets are maintained in a traceable workspace. The successful integration of the developed eFMI tooling is demonstrated by a comprehensive open source Modelica test cases library and industrial demonstrators. The readiness of the proposed approach is proven by compliance checks according to common automotive code quality standards like MISRA C:2012 and a performance benchmark in terms of runtime and resource demand in comparison with state-of-the-art hand coded solutions.

Keywords: embedded software, model-based development, code generation, model exchange, Modelica, FMI, eFMI, GALEC

1 Introduction

1.1 Motivation

Software has become an innovation driver not only but especially in the automotive industry. This has been leading to new challenges in terms of maintainability of the growing software stack for a growing number of ECUs (Electronic Control Units) in vehicles.

In the field of application software, it is the growing variability of the vehicles, increasingly demanding regulations and new powertrain solutions that add to the complexity of control and diagnosis functions. Original equipment manufacturers (OEM) as well as Tier 1 suppliers are aiming to cope with these demands by using mathematical models of physical systems (referred to as physical models) as part of the control software. For example, in Zimmermann et al. (2015) physical models are considered essential to manage the growing complexity of Diesel engine control as map-based approaches lead to an overwhelming calibration effort to satisfy the requirements of real driving emissions (RDE). Englert et al. (2019) present a framework for embedded non-linear model predictive controllers (NMPC) as a very generic approach of integrating physical models into a controller. With the increased computational power of modern multi-core ECUs, like the Bosch MDG1 (Rüger et al. 2014), these advanced approaches become relevant for industrial applications. In addition, e.g. Bosch is holding patents on methods for real-time applications of physical models (e.g. Wagner et al. 2009) stressing the fact that managing this type of applications is a differentiating selling proposition.

Model-based development (MBD) is an established paradigm for the development of control software for embedded targets deemed to ease these challenges. In practice the commonly used signal flow-oriented models, restricted to a predefined set of blocks, do not scale to the need. The models are rather a model of the software, than a model of the physical system with poor means to reflect variants of the underlying physical structure. The rather low-level description requires a high level of expertise of the modeler far beyond the physical behavior of the system including floating-point arithmetic, embedded software regulations, e.g. MISRA C:2012 rules (MISRA 2013-03) and target architectures, e.g. the AUTomotive Open System Architecture (AUTOSAR) (AUTOSAR Consortium 2021).

Despite valuable pioneering work on FMI (Functional Mock-up Interface) (Modelica Association 2021-04) in AUTOSAR, discussed in the following section, there is today no simulation solution supporting the export of physical models suitable for direct integration into controls, and no standard of any kind supporting efficient generation and integration processes from physical models to embedded software.

The goal of FMI for embedded systems (eFMI) is to overcome the known limitations of FMI, to enable new ways of model-based development of embedded software functions for arbitrary targets and architectures, based on advanced physical models of the underlying system.

1.2 State of the art of FMI in AUTOSAR

In a case study by Bertsch et al. (2015) it has been demonstrated that it is possible to wrap the C code from a Source Code FMU (Functional Mock-up Unit), that has been generated from a Modelica (Modelica Association 2021-02) physical model, as AUTOSAR ASW-C (Application Software Component) and to execute it on a Bosch ECU. This study also revealed the conceptual weaknesses of FMI when it comes to embedded automotive software and safety critical applications.

The probably most obvious weakness roots in the purposeful design of FMI being a standardized interface without being directive about the implementation of interface functions, except of which entry level files to provide and which headers to include. In contrast to that, the widely accepted rules of the Motor Industry Software Reliability Association (MISRA) are very specific about how a function is to be implemented in the C language to avoid ambiguities and runtime exceptions, while ensuring readability of the code. The goals of maximum flexibility for the exporting tool to produce any kind of *simulation code* to be wrapped into a black box running on a personal computer (PC) vs. high quality embedded *production code* being subject to thorough code reviews before being deployed to a dedicated target according to a certified procedure are contradicting. FMI is not designed to provide production code and binaries that are ready to satisfy the requirements of automotive embedded software quality gates. There are no mechanisms for the traceability of the code and no attributes about the used compilers and compiler settings to ensure the repeatability of the build process of a binary targeting a dedicated application in a specific runtime environment with the type of microcontroller already defined.

The prototypical tooling developed by Bertsch et al. (2015) enhanced by Neudorfer et al. (2017) is focusing on the technical aspect of translating a Source Code FMU into an AUTOAR SW-C. The aspects of which meta data has to be provided to support an end-to-end tool chain from the original model to the deployed binary is not discussed and remains as an unresolved issue of the prototypical work not intended for productive usage.

In terms of code quality Bertsch et al. (2015) state that none of the inspected source codes of the evaluated tools fulfilled their requirements and that the “C-code from many commercial tools is not suitable to run on an ECU due to its size and complexity since it was not intended to run on an embedded system”.

The tooling presented by Bertsch et al. (2015) translates in a first step the FMI `modelDescription.xml` file into a corresponding AUTOSAR `.arxml` file, based on a number of design decisions made on the desired representation as SW-C. The second step involves the translation of the C code of the Source Code FMU into C code that can be processed by build tools from Bosch for engine control software. After inspection of the source code generated by three different Modelica tools certain patterns were identified, such as: moving declarations to public or private headers, exclusion of functions from, e.g., `stdio.h` and `math.h` (ISO/IEC 2018-06) which are not supported on embedded devices, taking care of proper assignment of float values and avoiding implicit type casts. This process had been automated to a large extent, but made assumptions on the structure of the code and was finally still relying on the expertise of an embedded software developer to inspect the translated code and to fix remaining issues before further compilation.

This involved procedure illustrates that, if the generator of the C code is not giving any guarantees on its code, then it is very difficult, if not impossible, for the consuming tool to enforce these afterwards. From a workflow perspective, it is also highly objectionable when only after importing and processing of an FMU deficits are revealed that have to be addressed by the exporting tool.

All these issues are only about just compiling the code; but to fulfill the requirements of an embedded software the following aspects regarding the behavior and resource demand of the code have to be addressed as well:

- Limited data memory and code memory.
- Limited computation power of the target.
- Limitations on supported data types: 32-bit vs. 64-bit floating-point precision, fixed-point arithmetic etc.
- Static memory allocation only.
- Guaranteed exception freeness: Programs never fail due, e.g., unavailable/busy external devices, writing read-only memory, accessing multi-dimensions out-of-bounds, running out of stack memory etc.
- Guaranteed execution time within the limits of the available target system resources and the minimal sampling period required for correct physical behavior.
- Proper error handling: Handling of Not a Number (NaN) (IEEE 2019-07), mathematical functions that are undefined for some arguments, linear systems without unique solution etc.
- In bound guarantees of signals.

Simulation code generated by today’s FMU generating tools is optimized for runtime performance on a PC. The memory required by the FMU is allocated dynamically. The sizes of the data and program code are not considered limiting factors and therefore not minimized. This renders existing FMI solutions unsuited for application in the embedded domain.

For co-simulation FMUs (CS-FMU) there are no restrictions for the type of solver being used. For real-time applications, variable-step-size solvers are not suitable, as they cannot guarantee the execution time on an equidistant time grid. Event handling and non-linear algebraic loops are particularly challenging, since self-evident unbounded iterative solutions of such cannot be used; instead, such have to be expressed as upper-bounded iterative algorithms with execution time guarantees. If this is not possible, the system is not suited for embedded real-time.

Hence, the consumer of an FMU fully depends on the modeler and exporting tool to have made appropriate choices in setting up the model and making the settings for the solver and the compiler to meet the basic runtime requirements. An FMU by itself gives no guarantees further processing can rely upon.

Furthermore, FMUs are expected to run in a simulation or co-simulation environment that provides exception handling mechanisms. Messages may be dumped into a log file in case of exceeded value ranges based on the assert statements in the code for the simulation engineers to verify that they can trust the results. In safety critical applications, there is no second attempt. The software must guarantee exception free execution and a predictable behavior under all circumstances. FMI does not provide any means to cope with this requirement.

Despite the fact that one can find ways to translate the C code from a Source Code FMU to compile and being executed on an embedded target, one has to state that FMI has no means to guarantee that the source code fulfills basic prerequisites for embedded real-time execution, nor does it provide a rich enough model description to support an end-to-end build chain in terms of traceability, transparency and repeatability of the process.

1.3 eFMI vs. FMI

eFMI helps to overcome the shortcomings of FMI, explained in Section 1.2, for the development of embedded software based on physical models. eFMI is not just an extension of FMI; it is an orthogonal, new standard that is maintained and further developed in a separate Modelica Association Project. Entirely new concepts are introduced, but at the same time, a high degree of consistency with FMI has been achieved. Whereas FMUs are ready-made "consumer" products for exchanging simulation models, an *FMU for embedded systems* (eFMU) is a shared development workspace for step-wise, semi- and full-automatized refinement from a high-level intermediate representation of a sampled algorithm (in the GALEC language, see Section 2.4) to an implementation of the algorithm for an embedded target. The development of a single eFMU is shared between varying eFMI tools and developers. Throughout its development, the eFMU very likely is in intermediate stages not suited for simulation. The developed final solution can be wrapped by a respective FMI interface such that it behaves like a regular FMU. Doing so, the outer FMI shell allows any FMI supporting tool to load and execute the eFMU as CS-FMU, accessing the actual production code through appropriate wrappers. This enables verification and validation (V&V) of the target code in a Software-in-the-Loop (SiL) environment without restricting the target code to satisfy a static C interface.

1.4 eFMI Workflow

Key differentiator of the proposed eFMI workflow (see Figure 1) is a multistep approach reflected by different types of so-called *model representations* that are stored in containers within the same eFMU. Each model representation addresses a different aspect and refinement step of a physical model to embedded software in the development process. The intention is to provide an automatable workflow, where the model representations can be generated, with tool-supported refinement along the "*Transform*" boxes in Figure 1.

The *Algorithm Code* model representation (see Section 2.4) provides a target independent, intermediate representation for upper-bounded algorithmic

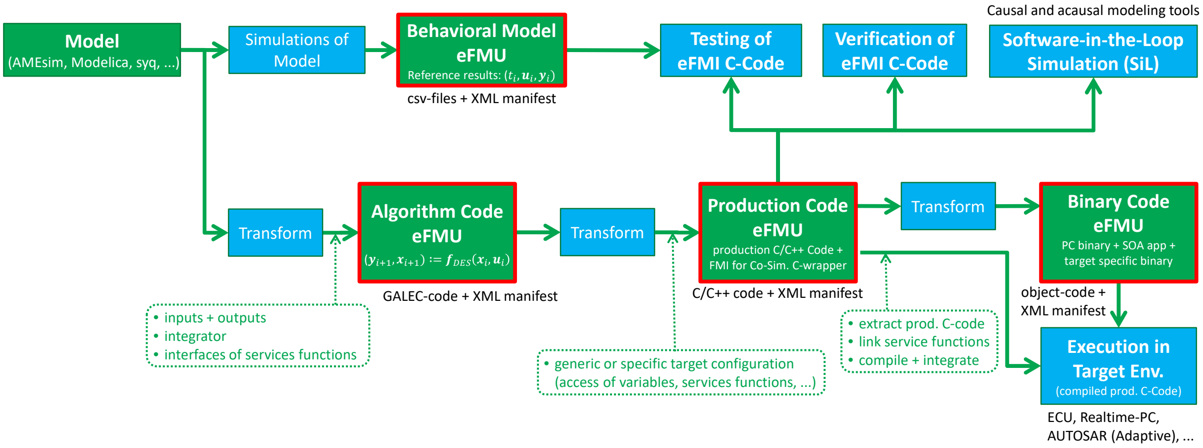

Causal and acausal modeling tools
Model
(AMEsim, Modelica, syq, ...)
Simulations of Model
Behavioral Model eFMU
Reference results: $(t_i, \boldsymbol{u}_i, \boldsymbol{y}_i)$
csv-files + XML manifest
Testing of eFMI C-Code
Verification of eFMI C-Code
Software-in-the-Loop Simulation (SiL)
Transform
Algorithm Code eFMU
$(\boldsymbol{y}_{i+1}, \boldsymbol{x}_{i+1}) := \boldsymbol{f}_{DES}(\boldsymbol{x}_i, \boldsymbol{u}_i)$
GALEC-code + XML manifest
Transform
Production Code eFMU
production C/C++ Code + FMI for Co-Sim. C-wrapper
C/C++ code + XML manifest
Transform
Binary Code eFMU
PC binary + SOA app + target specific binary
object-code + XML manifest
Execution in Target Env.
(compiled prod. C-Code)
ECU, Realtime-PC, AUTOSAR (Adaptive), ...
• inputs + outputs
• integrator
• interfaces of services functions
• generic or specific target configuration (access of variables, services functions, ...)
• extract prod. C-code
• link service functions
• compile + integrate

Figure 1. eFMI workflow with its different model representations (red boxes).

computations. It serves as code generation target for (physics-based) modeling tools, allowing them to concentrate on the general concern of finding a sequence of computations that satisfies real-time constraints, i.e., is algorithmically well-defined with an upper-bound of computational steps.

Such algorithmic solutions are further refined to actual implementation code that is best suited for a specific target environment in terms of performance, memory consumption, software architecture and applicable rules and regulations by *Production Code* model representations (see Section 2.5). For a single Algorithm Code container, several Production Code containers can be given to address varying target platforms, e.g., alternative chip sets or customer specific code guidelines.

For each production code, arbitrarily many compiled binaries, tailored for embedded system integration within a specific ECU/target platform, can be included via *Binary Code* model representations (see Section 2.6). Besides defining a target specific build and integration, Binary Code containers can also protect intellectual property by only providing the binaries as such without sources.

Finally, validation and verification (V&V) is covered by means of *Behavioral Model* model representations (see Section 2.3), which allow to store reference results for later back-to-back testing of other model representations like Production and Binary Code containers.

Starting from an Algorithm Code container, there is no further stringent order in which traceable containers must be provided or updated throughout eFMU development iterations. This enables full flexibility in the software development process supporting all kinds of model and software sharing schemes for OEMs and suppliers, with eFMI as open standard giving maximum freedom in the choice of tools. An eFMU is a standardized workspace for collaborative development of embedded solutions from (physical) models.

1.5 Structure of the paper

This paper gives a coarse introduction to the basic concepts of the eFMI Standard (Section 2) and highlights the achieved goals in terms of readiness and applicability to industrial grade problems (Sections 3, 4), before summarizing the future work and the conclusions (Sections 5, 6).

2 eFMI Standard

The following description of the eFMI Standard is according to version 1.0.0-alpha.4 (EMPHYSIS 2021-07) published in February 2021; on the same web page an example eFMU can be downloaded.

2.1 Mathematical description of eFMI

As described in Section 2.6, the starting point of eFMI workflows are typically physical models for some independent modeling and simulation environment, e.g., a Modelica tooling (Modelica Association 2021-02). Such original models can be described by (unsorted) equations, algorithms or functions, which have to be transformed to eFMI Algorithm Code model representations. This requires a causal, discretized algorithmic solution to be found for the acausal physics equations – hence the modeling environment must provide a GALEC code generation backend (see Section 2.6).

Mathematically, an algorithmic solution can be described as a *sampled input/output block* with *one* (potentially varying) sample period $T_i = t_{i+1} - t_i$ for the whole block. Inputs $\boldsymbol{u}_i = \boldsymbol{u}(t_i)$ and previous block internal states $\boldsymbol{x}_i$ are provided at sample time t_i whereas outputs $\boldsymbol{y}_i = \boldsymbol{y}(t_i)$ and new states $\boldsymbol{x}_{i+1}$are computed in the block (see Figure 2).

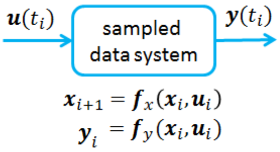

$\boldsymbol{u}(t_i)$
sampled data system
$\boldsymbol{y}(t_i)$
$\boldsymbol{x}_{i+1} = \boldsymbol{f}_x(\boldsymbol{x}_i, \boldsymbol{u}_i)$
$\boldsymbol{y}_i = \boldsymbol{f}_y(\boldsymbol{x}_i, \boldsymbol{u}_i)$

Figure 2. Mathematical description of an algorithmic solution as supported by eFMI Algorithm Code model representations and the GALEC language.

All variables of the block have a defined type and all statements of the block are sorted and explicitly solved for a particular variable. Functions are provided to execute the relevant parts of the block, especially to initialize it (`Startup()` function) and to perform one step (`DoStep()` function).

To find an upper-bounded algorithmic solution for acausal equation systems is a non-trivial task, with reasonable solution strategies highly depending on characteristics of the original modeling language. The eFMI Standard therefore is not covering, or in any way restricting on how to find suited solutions; it solely concentrates on defining how such solutions must look like (see Section 2.4) to be processable by further eFMI tooling like production code generators.

2.2 eFMU Container Architecture

The basic file structure of eFMUs is:

```
/eFMU ; eFMU root directory
   /<directories> ; Model representations
   /schemas ; eFMI XML Schema definitions
   __content.xml ; eFMU content-manifest
```

The only mandatory file is the eFMU content-manifest (`__content.xml`) at the root of the `eFMU` folder. In addition, an eFMU includes the XML Schema files defined by the eFMI Standard (`/schemas`) to become self-contained; these XML Schemas restrict the structure and content of the manifests of the varying eFMI model representations and thereby enable automatic processing of eFMUs and their content. All other directory and file names within an eFMU can be freely chosen by the tools generating and processing the varying eFMI model representations. The directories containing some model representation, like an Algorithm Code or Production Code container, are denoted in the eFMU content-manifest, which also lists the type of model representation and other meta information like checksums. Each model representation further supplies its own manifest according to its type of representation; manifests of model representations typically list all files of the representation, their checksums and other representation specific meta information, like the in- and outputs of GALEC programs (Algorithm Code containers) or the compiler settings used to produce some binary code (Binary Code containers).

The structure of a typical eFMU could look like this:

```
/eFMU
   /BehavioralModel
      manifest.xml ; Container manifest
      <other files>
   /AlgorithmCode
      manifest.xml ; Container manifest
      <other files>
   /ProductionCode_Generic_C_Float32
      manifest.xml ; Container manifest
      <other files>
   /ProductionCode_Generic_C_Float64
      manifest.xml ; Container manifest
      <other files>
   /ProductionCode_Autosar_Float32
      manifest.xml ; Container manifest
      <other files>
   /schemas
   __content.xml ; eFMU content-manifest
```

An eFMU can be packed in different formats.

1. The eFMU root directory is a standard directory in the file system. This is useful to hold an eFMU in a text-based version control system, such as git or SVN.
2. The eFMU root directory is zipped with the eFMU-content and is stored in a zip-file with the extension `.efmu`. This is useful to ship or distribute an eFMU.
3. The eFMU root directory is path `extra/org.efmi-standard` inside a standard FMU. The path is defined according to the current FMI-3.0-beta.1 pre-release of the FMI specification (Modelica Association 2021-04). With attribute `activeFMU` inside the eFMU content-manifest it is defined which of the Algorithm, Production or Binary Code representations is used as basis of the FMU. This package format is useful to ship or distribute an eFMU for Model/Software/Hardware-in-the-Loop (MiL/SiL/HiL) simulation by further FMU tooling.

Note, that Algorithm Code, Production Code and Binary Code representations can optionally store associated FMUs. For example, Algorithm Code representations can store a Model-in-the-Loop FMU and Production Code representations for different targets can store Software-in-the-Loop FMUs. To execute these FMUs, they must be extracted from the respective model representation – manually or by a tool. If an eFMU is organized according to package format (3), a selected FMU from a model representation has to be copied to the root level so that the eFMU behaves as an ordinary FMU and can be simulated by any FMI tool.

2.3 Behavioral Model Representation

The Behavioral Model representation of eFMI is designed to describe functional and behavioral aspects of the original (physical) model/system/controller with the goal of enabling the validation of other generated model representations within the same eFMU. The central question is how to define and include reference behavior for varying model representations and their respective software? As an example, one can think about a controller model in Modelica represented by ordinary or differential algebraic equation systems that shall be exported as eFMU. Reference result data may be important for several use case scenarios, e.g. simulation runs of the controller model in a Modelica tool:

- with a complex variable step-size integration algorithm to define a highly accurate reference solution,
- with a fixed step-size algorithm like the Explicit Euler method and a fixed step-size to get a reference solution of the discretized sampled data system,
- in different model setups like open and/or closed loop scenarios to cover a broad range of possible input value combinations,
- in other test scenarios to test specific features/requirements of the controller – like unit tests in software development – and
- all the tests may be run using the floating-point precision typically used in offline simulations of 64-bit or/and the more common precision for embedded systems of 32-bit.

To structure this variety of possible reference data a rather simple format with only two hierarchies (scenarios and scenario parts) has been designed: An eFMI Behavioral Model representation consists of scenario parts grouped in one or more use case scenarios. Each scenario part represents a single behavioral aspect of the original model given by different time dependent input/output data in a comma-separated values (CSV) file, e.g. a closed loop scenario with an Explicit Euler discretization of the controller model. Absolute and relative error tolerances or time dependent lower and upper bounds can be defined for each variable in the Behavioral Model manifest, to account for deviations resulting from discretization methods, implementation data types and floating-point imprecision.

Since variable names as well as data types might differ between the different eFMI representations within the same eFMU (e.g., different production code variants might map GALEC variables to different C identifiers to satisfy naming conventions of varying target environments), every variable of a Behavioral Model container is linked with its corresponding variable of the Algorithm Code container such that reference result data can be provided easily and fully automatic for each model representation. This means for example, that later added Production Code containers can be automatically tested using existing Behavioral Model containers thanks to the trace-links between variables of the manifests of Behavioral Model and Production Code containers to the manifest of the Algorithm Code container.

A typical validation approach consists of the execution of compiled eFMI production or binary code with input data from the CSV files. These simulation results are compared with the expected output data of the Behavioral Model representation according to the given error tolerances or bounds to validate the contained eFMI representations against the "behavior" of the source of the generated eFMU. This approach enables tools (such as testing tools) to perform a fully automatic validation of other eFMI representations (Algorithm Code, Production Code and Binary Code representations) within an eFMU w.r.t. behavioral and functional equivalence to the physical model using a back-to-back testing approach.

2.4 Algorithm Code Model Representation

All containers in an eFMU are related to the Algorithm Code container of which exists exactly one in each eFMU. It provides a high-level intermediate representation of a sampled algorithm by means of a GALEC block and a description of the block's interface by means of an XML manifest. The manifest is used by other containers to trace their dependencies on the block and avoid processing GALEC programs if just in need of a description of the block interface. The GALEC block can define any kind of finite sampled computation that is subject to embedded integration (controller, virtual sensor etc.). In terms of the Modelica language, a GALEC block is a causal solution for the sampled system defined by a clocked partition.

GALEC is a new imperative programming language part of the eFMI Standard. Its name is an abbreviation for ***g***uarded ***a***lgorithmic ***l***anguage for ***e***mbedded ***c***ontrol. It is an intermediate representation between the (physics) modeling and embedded programming domains.

GALEC Characteristics: The language characteristics C1-11 of GALEC, making it a suitable intermediate representation between modeling and embedded software, are:

(C1) Target independence: Arithmetic and algorithms are on an ideal machine, with built-in functions for abstract handling of target dependent operations like retrieving the fraction part of a real or checking if such is Not a Number (NaN) (IEEE 2019-07).

(C2) Explicit language semantics: No implicit casts between integer and real, no hidden side effects and no default arguments; simple name space without shadowing, overloading or polymorphic functions. These restrictions are kind of language-enforced MISRA C:2012 rules.

(C3) Multi-dimensional arithmetic: Support for vectors, matrices and higher dimensions with respective scalar and matrix multiplication, addition etc. Production code generators can leverage on (C4) to map multi-dimensional operations to efficient implementations, for example Streaming SIMD Extensions 4 (SSE4) machine instructions (Intel Corporation 2021-06).

(C4) Powerful static evaluation: GALEC expressions are separated into three kinds: (1) declarative sizes, (2) algorithmic indexing and (3) algorithmic runtime computations, with the former two being subject to static evaluation for mandatory well-formedness analyses. Algorithmic indexing hereby includes `for`-loop iterators and static evaluation of their ranges. Indexing expressions can depend on loop iterators but must not refer to any other variables for their value. Otherwise, statically evaluated expressions can be arbitrary complex, including calls of built-in functions.

(C5) Upper-bounded: GALEC programs must be non-recursive; the only iteration construct are `for`-loops, which according to (C4) can be unrolled. Thus, every program can be unfolded to an iteration-free sequence of conditional assignments defining an upper bound of algorithmic steps. This characteristic enables worst time execution analyses and advanced optimizations.

(C6) Computational-safe: The static unfolding-characteristics of (C4) and (C5) are used to guarantee all indexing is within bounds. Production code generators can avoid dynamic memory allocation, optimize the memory mapping and eventually ensure a target's resources are always sufficient for exception-free program execution.

(C7) Control-flow integrated error signal handling: Language constructs to signal errors and handle signaled errors using ordinary control-flow conditions. All potentially signaled errors must be handled or explicitly exposed to the runtime environment; they cannot slip through unnoticed. Automatic error signal propagation enables delayed error handling, avoiding the need for immediate checks of each operation that might fail.

(C8) Safe floating-point numerics: Guaranteed quiet NaN and infinity propagation according to IEEE 754-2019 (IEEE 2019-07), with relational operations signaling pre-defined errors when called with NaN arguments. Such integration with the error signaling concept of (C7) means, that NaNs can never slip through unnoticed.

(C9) Safe built-in functions: Rich set of safe built-in functions for casting, numeric limits, rounding, trigonometric operations, 1/2/3D interpolation, solving systems of linear equations etc. If arguments are out of

range, the error signaling of (C7) is used to denote so and returned values are precisely defined (typically NaN or infinity) to avoid undefined, implementation dependent, behavior; in line with (C8), NaN arguments are preferably silently propagated.

(C10) Call-by-value semantic with well-defined side effects: Function arguments are passed by value; and, although an imperative language, GALEC has well-defined side-effect rules that guarantee (1) each expression is free of competing side effects and (2) which statements are mutual independent. Production code generators can leverage on these characteristics for automatic, lock-free program parallelization and to avoid unnecessary copying of multi-dimensional values.

(C11) Block life-cycle with well-defined layers of modification: GALEC supports a layered modification concept distinguishing constants from semi-constant tunable parameters from dependent parameters, parameters from block in- and outputs and such from inner states. Only tunable parameters and inputs can be directly changed by the runtime environment, but only in-between two sampling steps, never while sampling. Whenever tunable parameters are changed, dependent parameters must be recomputed based on the new tunable parameters only (`Recalibrate()` interface function); dependencies on states, in- or outputs are forbidden. In addition, initialization is clearly encapsulated (`Startup()` interface function) with mandatory data-flow analyses guaranteeing every block variable is assigned an initial value based only on literal values or already initialized variables. Initialization code can contain arbitrary complex algorithms; its clear encapsulation enables static evaluation. The actual sampling code, which computes new outputs for given inputs considering the current block state, is encapsulated in the `DoStep()` interface function; its implementation must not change inputs nor parameters. All block interface functions automatically saturate variables with declared ranges (ranged variables) at the very beginning and ending of their execution. This guarantees, for example, that inputs and outputs are always within their ranges when `DoStep()` starts and terminates, yielding the behavior of a saturated controller. Not only block interface variables are saturated, but also inner states if respectively ranged, or ranged parameters when recalibrating. The whole block life cycle is formally defined via a state machine.

For details, readers are encouraged to consult the public alpha draft of the eFMI specification (EMPHYSIS 2021-07).

GALEC Example: To give at least a glimpse on how GALEC programs look like, particularly error handling, a short artificial example is given in the following. A typical GALEC block looks like the following (`[[...]]` denotes removed code snippets):

```
block Controller
  // Block interface variables:
  input  Real u[10] (min = -1.5, max = 1.5);
  output Real y[20] (min = -1.0, max = 1.0);
  parameter Real tP;    // tunable parameter
  parameter Real tV[20];// tunable parameter

protected
  // Internal block variables and functions:
  parameter Real dP; // dependent parameter
  Real M1[20,10]     // state
    (min = -1.0, max = 1.0);
  Real M2[10,20]     // state
    (min = -1.0, max = 1.0);
  function checked_transpose
    signals UNDERFLOW, NAN [[...]];
  function sum [[...]];

public
  // Block interface functions:
  method Recalibrate
    signals INVALID_ARGUMENT [[...]];
  method Startup [[...]];
  method DoStep
    signals NO_SOLUTION_FOUND [[...]];
end Controller;
```

First, the block interface variables that can be set (inputs and tunable parameters) and read (outputs) by the runtime environment are declared. Like any variable, such can be multi-dimensions and ranged. E.g., `y` is an output vector of size 20, with each of its elements in the range [-1.0, 1.0]. Then the section with the internal block variables and functions follows, first the dependent parameters, then the states and finally functions. Note, that any errors a function can signal to callees are part of its interface. `checked_transpose` for example can signal `UNDERFLOW` and `NAN` error signals. Finally, the section with the block interface functions follows. These are `Recalibrate()`, `Startup()` and `DoStep()`. Note, that in the example, a sampling step can signal that no solution has been found via the `NO_SOLUTION_FOUND` signal; the signal is used in the example to denote that a fallback controller has been used due to an unexpected error. The `INVALID_ARGUMENT` of the `Recalibrate()` function is used to denote to the runtime environment that given new tunable parameters are invalid and another recalibration is required. Of course, these error signals are just examples; the interface functions of other blocks may signal different, or no errors at all.

Assume `checked_transpose` is defined as follows:

```
function checked_transpose
  signals UNDERFLOW, NAN;
  input Real In[:, :];
  output Real Out[size(In, 2), size(In, 1)];
algorithm
  for i in 1 : size(I, 1) loop
    for j in 1 : size(I, 2) loop
      Out[j, i] := In[i, j];
      // Signals NAN if any argument is NAN:
```

```
      if absolute(In[i, j]) <
        epsReal() * self.dP
      then
        signal UNDERFLOW;
      end if;
    end for;
  end for;
end checked_transpose;
```

Its in- and output are any matrices of reversed dimensionality. If used in a context where the output is not an $n \times m$ matrix for an $m \times n$ input, static dimensionality analyses will fail with an error (cf. (C6)). The `for`-loop uses the matrix dimensions to traverse all elements; it computes the transpose of `In` in `Out`. Thereby every element is checked to be non-zero around an epsilon based on the target machine's minimal precision (`epsReal()` built-in function) and the block's dependent parameter `dP`; the latter is accessed directly via `self.dP`. If the check fails, `UNDERFLOW` is signaled. The `<` check itself will signal `NAN` if any of its arguments is NaN. This behavior is guaranteed by GALEC (cf. (C8)). Since neither of both signals is handled within `checked_transpose`, both must be exposed to callees as denoted in the function's interface (the `signals UNDERFLOW, NAN;` following the function name).

Assume the sampling function is:

```
method DoStep
  signals NO_SOLUTION_FOUND;
algorithm
  self.M1 := sum(self.u) /
    real(size(self.u, 1)) * self.M1;
  self.y := solveLinearEquations(
    self.dP * self.M1 * self.M2,
    self.tV);
  self.M2 := checked_transpose(self.M1);
  // Catch any error signals
  // or NaN/∞ in self.y:
  if signal or not(allFinite(self.y)) then
    [[ ...fallback controller code... ]]
    // Expose use of fallback controller:
    signal NO_SOLUTION_FOUND;
  end if;
end DoStep;
```

At the very end of all computations, a simple conditional control-flow checks for any kind of errors, and in case of any error, uses some fallback controller and signals its usage by exposing the `NO_SOLUTION_FOUND` signal to the runtime environment. This delayed error handling is achieved by the conditional:

```
 if signal or not(allFinite(self.y))
```

The `if signal` construct can be used to check for any, only specific or any except certain error signals (`if signal`, `if signal in` E_1, E_2, …, E_n and `if signal not in` E_1, E_2, …, E_n respectively). The body of the check is executed if any of the checked signals was set; if so, the signals are automatically unset. In the example's case, all error signals are handled. The `or` condition is optional; it is used in the example to check if any value of the block's output vector `y` is NaN or +/-∞ via the `allFinite` built-in function. Error signal checks are ordinary control-flow conditionals and can be combined with any other `if`, `elseif` and `else` conditioned branches. In the example, errors might be signaled by the `checked_transpose` call or the `solveLinearEquations` call. The latter built-in function fails with a predefined error if the linear equation system `Ax = b` cannot be solved, with `A` being its first argument, `b` the second and `x` its result. Note, that in the example, the first argument is computed using multi-dimensional arithmetics: the scalar dependent parameter `dP` is multiplied to the 20 × 10 matrix `M1`, the resulting 20 × 10 matrix in turn is multiplied to the 10 × 20 matrix `M2` yielding a quadratic 20 × 20 `A` matrix as required by `solveLinearEquations`. Finally, `DoStep()` will according to (C11) implicitly, at the very end, saturate the block output `y` and all elements of matrices `M1` and `M2` to be in the range [-1.0, 1.0], as it will implicitly saturate the block input `u` to be in range [-1.5, 1.5] at its very beginning (since these block variables are declared with ranges). Of course, only non-NaN values can be saturated; NaNs stay.

Tool challenges: A Modelica tool (or any other modeling tool) targeting GALEC for code generation can concentrate on the actual computation by leveraging on its high-level abstractions for multi-dimensional arithmetic, whereas embedded tooling benefits from the inherent language guarantees every GALEC program will satisfy and these are of uttermost importance for embedded software (like guaranteed termination with upper-bound of algorithmic steps or exception-freeness). Of course, a major challenge for Modelica tools is to actually find an upper-bounded causal solution for a given acausal equation system; this is a non-trivial task with many challenges on developing suited integration schemes. Once developed, respective approaches are however naturally/conveniently expressed as GALEC programs.

2.5 Production Code Model Representation

The previously described formal representation of a control algorithm in the GALEC language must be transformed into executable code for a target machine. For the generation of this code (production code) there are many degrees of freedom. In contrast to FMI, eFMI does not enforce a strict code and API format but allows the actual Production Code representation of an algorithm to be adjusted to the context in which the code is to be integrated into. The container architecture of an eFMU allows to hold several such Production Code model representations. An integrator of production code can pick the one that is suitable for his integration context.

The integration context determines several characteristics including the available bit size (e.g. integer as well as single or double precision floating-point), different data interfaces (e.g. functions without arguments working on global data vs. functions with arguments), target and compiler specific optimization, as well as completely different platforms (such as e.g. AUTOSAR vs. plain C code). The different production codes for the same GALEC program and their integration contexts are described in their respective manifest. Each manifest contains all information to enable an integration of its production code into a test or execution environment.

We would like to illustrate the impact of the integration context onto the actual production code by giving alternative C18 (ISO/IEC 2018-06) code realizations for the GALEC example of Section 2.4. In the first "software architecture", all block variables are realized as global variables:

```
float u[10];
float y[20];

float tP;
float tV[20];
float dp;
float M1[20][10];
float M2[10][20];

unsigned int DoStep(void)
{
  unsigned int signals = 0U;
  [[...]]
  return signals; /* NO_SOLUTION_FOUND? */
}
```

In the second architecture below, the input-output notion of the block is mapped to an input/output of the `DoStep` function itself and the block state (parameters and inner states) is encapsulated in a passed `struct` which must be allocated by the embedded runtime environment (note that the block state and output are passed by reference, thus are writeable by `DoStep`):

```
typedef struct
{
  unsigned int signals;
  float tP;
  float tV[20];
  float dp;
  float M1[20][10];
  float M2[10][20];
} BlockState;

void DoStep(
  BlockState* const state,
  const float const u[10],
  const float y[20])
{
  state->signals = 0U;
  [[...]]
}
```

This scheme allows multiple, independent instances of the block to be allocated by the runtime environment (`DoStep` itself is stateless).

A third architecture could be the AUTOSAR Classic Platform. Here, variables are accessed using macros provided by a centrally generated middleware.

Note, that independent of the software architecture, many characteristics of GALEC (e.g. no need for dynamic memory allocation, bounded execution time with bounded loop iterations) are intrinsically also properties of derived production code.

Other characteristics such as the handling of error signals or the support of multi-dimensional arithmetic let more room for production code generating tools to exploit different solution alternatives and are not straightforward. For multi-dimensional arithmetic, specific libraries could be included, and for error signal handling a low-level mapping using bit masking logic can be performed in the transformation process from GALEC to production code.

The representation of error signals using bit-masking logic, for example, allows for fast (simultaneous) check of several conditions and the efficient setting and resetting of all concerned signal values. A GALEC snippet like

```
if signal in OVERFLOW, NAN then
  y := y + 1.0;
end if;
```

could be translated into C18 production code like

```
if ((signals & 0x6U) != 0U)
{
  /* First, reset the checked signals: */
  signals = signals & 0xfffffff9U;
  /* Then, proceed with body: */
  y = y + 1.0F;
}
```

with proper encoding of the signal values for `OVERFLOW` and `NAN` in bit positions 2 and 3.

For multi-dimensional arithmetic and interpolation routines, usually libraries optimized for the target platform will be used. The production code generator has to make sure that used data structures of matrices and vectors match the format of the used libraries. Furthermore, the production code generator has to take care that the value-semantic of the GALEC language is preserved by taking adequate precaution like copying data when using library algorithms that alter the input data (e.g. when solving linear systems). Data flow analysis on the GALEC program enables optimizations that can avoid unneeded copy operations. Operations that are "simple/atomic" in GALEC code (like chained arithmetic expressions on multidimensional elements) may require a "flattening" in the generated production code and a proper non-trivial management of intermediate results.

In the example of Section 2.4, the multi-dimensional arithmetic expression

```
self.y := solveLinearEquations(
  self.dP * self.M1 * self.M2,
  self.tV)
```

for example requires to store the intermediate result of multiplying the scalar `dP` with the 20 × 10 matrix `M1`; the resulting 20 × 10 matrix has to be multiplied with the 10 × 20 matrix `M2`, yielding a temporary 20 × 20 matrix which is passed as `A` argument to `solveLinearEquations`.

Other optimizations on the production code generator side include analysis of value ranges to be able to omit unnecessary saturation operations for in-, outputs and states in case it can be determined that the values are always within bounds.

Another aspect of the production code generation step is to make the generated production code accessible and interpretable by consumers of it in subsequent phases like testing or integrating into an ECU SW. With the large degrees of freedom in generating production code for a given GALEC program, the structure of the production code may vary greatly, but must be described unambiguously at least in the interface parts and must be made accessible to anyone who would like to interact with it. This is made possible by the Production Code manifest, which precisely describes the code structure and interfaces (e.g. types, variables, functions) as well as their association to the corresponding elements of the GALEC code. This association is important to map information available only on the GALEC level also to the production code elements and enable traceability of the multi-step generation process. For example, stimulation data in a Behavioral Model container that is mapped to GALEC block variables can be applied also to their counterparts in the production code, or attributes of these variables (like ranges, units) can be associated to their respective production code counterparts. The required cross-referencing between different eFMU containers, e.g., to the manifest of the Algorithm Code container, uses a unified referencing scheme.

Besides the integration interface, Production Code manifests give, for example, a precise description of the target (like target language, target platform, target type, compiler and linker options) and the code files that make up the production code. The content of such code files is described in terms of XML elements and attributes like *Includes*, *TypeDefs*, *Macros*, *Variables* and *Functions* with *FormalParameter* and *ReturnParameter,* including both a mapping to target specific realizations (e.g. target types) as well as a "backward" reference to the corresponding elements in the Algorithm Code container.

With the help of the meta information of Production Code manifests, other widely used standards like AUTOSAR and FMI can be supported. In case of an AUTOSAR platform, the code files are complemented with specific description files that contain all information to integrate the production code w.r.t. the used AUTOSAR standard. In case of the AUTOSAR Classic Platform for example, such description files are the `.arxml` files shipped with the software component.

2.6 Binary Code Model Representation

The eFMI Binary Code model representation contains binaries that have been derived from a Production Code representation for a dedicated target architecture. It mainly serves two purposes:

1. Support the creation (build process) and integration of binaries on an embedded ECU target.
2. Protect intellectual properties when software artifacts are shared in a collaborative development process with multiple parties.

The first purpose is achieved by providing (a) the actual binaries and (b) the relevant build information like compilation and linking steps to create these binaries for a certain target platform. (b) is done in the manifest of the respective Binary Code container and can also be used to rebuild the binaries in case they are stale due to later production code changes, whereas production code cross-referencing with mandatory checksums enables to automatically deduce if binaries are stale. Note, that existing compiler and linker information of production code manifests can be referenced and further refined by the manifests of Binary Code containers, enabling a stepwise specialization and dedication towards a target platform. Integrators can use the build information to integrate the binaries on their embedded target ECUs. Additionally, the manifest can list run time compliance information such as execution times and further information relevant for the integration (e.g., a calibration file describing memory addresses and value ranges for calibration).

Intellectual property protection is achieved by removing the source code of the Algorithm Code and Production Code containers a Binary Code container is derived from, such that they are left with their manifest files only. Doing so, binary implementations can be shared without exposing actual source codes to third parties, whereas the meta information required for embedded software integration are still provided by the manifests.

An example for the stepwise derivation of dedicated binaries is the AEBS demonstrator mentioned in Section 4.1, where a generic production code is refined with integration code for the AUTOSAR Adaptive Platform, from which eventually platform-specific binaries with accompanying AUTOSAR Adaptive Platform manifests are generated.

3 eFMI Readiness

3.1 eFMI Tool Support

The EMPHYSIS Consortium (EMPHYSIS 2021) with its 25 partners from Belgium, Canada, France, Germany and

Sweden covered the entire value chain from vendors of modeling and simulation tools, code generators and V&V tools over embedded software developers and integrators to automotive Tier 1 suppliers and OEMs. This allowed to develop the eFMI specification along with reference implementations that have been thoroughly tested (cf. Section 3.2) and applied to challenging industrial applications (cf. Section 4).

By the end of the project in February 2021, already 13 different tools covering the entire eFMI workflow plus the open source eFMI Compliance Checker were available as prototypes. Soon after the official release of the eFMI Standard, these tools are expected to be available on the market.

3.2 Test Cases and Coverage

A set of dedicated test cases has been extensively used for testing the eFMI workflow with implementations in different prototype tools during the EMPHYSIS project. Most of the test cases are part of the Modelica library *eFMI_TestCases* that has recently been published under a 3-Clause BSD license (Modelica Association 2021-07). A few other test cases are AMEsim models or manually implemented Algorithm Code containers. For each of the test cases automatically generated reference results are provided in respective Behavioral Model containers.

By altogether 48 test cases (including variants) the following partially very advanced features are covered: non-linear inverse models, feedback linearization based controllers, explicit and implicit integration schemes, event-based re-initialization of continuous states, neural networks, error handling, implicit saturation and important built-in functions like solving linear equation systems as well as 1-D and 2-D interpolation tables. Each feature is supported by at least one eFMI prototype tool generating Algorithm Code containers.

All generated Algorithm Code containers have been successfully imported by the involved production code tools. For each Algorithm Code container, two production code variants have been generated: A double precision floating-point (64-bit) and a single precision floating-point (32-bit) version, each with respective implementations of higher-level built-in functions.

A testing tool chain has been set up to automatically check all generated production code variants, create test harnesses, compile the code, execute it and compare the results with the reference results contained in the Behavioral Model containers of each eFMU with respect to given error tolerances. In total, 538 execution runs are necessary to assess all production code variants generated by the varying combination of tools along the eFMI workflow. More than 96% of these runs successfully passed. The unsuccessful tests are all a result of a currently incomplete initialization mechanism in one test case and its variations that will be the subject of investigation in future work. Nevertheless, the very positive test rate impressively shows the maturity of the tool prototypes and their compatibility.

3.3 Performance Benchmarks

Performance benchmarks of the generated production code against state of the art manually implemented C solutions have been conducted. The target was the Bosch Multicore ECU MDG1 (Rüger et al. 2014). Six test cases addressing known difficulties of physical models on ECUs by using automatic model to code transformations have been contributed to the *eFMI_TestCases* library. In the following, these test cases are denoted by their IDs in *eFMI_TestCases*; the addressed challenges, in ascending order w.r.t. difficulty, are:

- DC motor speed control with PID controller (*M03_B*): Minimal footprint of code with saturated inputs and outputs.
- Air system controller (*M15_A*): Stiff ODE with delay operator.
- Drivetrain torque controller based on inverse model (*M04_A*): Linear inverse physical model.
- Inverse slider crank (*M10_B*): Non-linear inverse physical model (DAE Index-1).
- Reduced order model of a thermal heat transfer (*M16_A*): Efficient handling of matrix operations and large two-dimensional maps.
- Ideal rectifier (*M14_A/B*): Advanced symbolic transformation to derive a compact state space form.

All models are tested in an open loop setup using the recorded data from their Behavioral Model container as stimulus. The execution time is captured based on the CPU ticks elapsed, right from the start of calling the model interface function (e.g. `DoStep`) until the function execution is completed (note, that the MDG1 ECU enables precise and reliable counting of elapsed CPU cycles without any caching effects). As they have a significant impact, boundary and error checks are considered. Boundary checks saturate the in- and outputs to their limit values. Error handling will check for non-plausible values like NaN and infinity. More details are provided in Armugham et al. (2021).

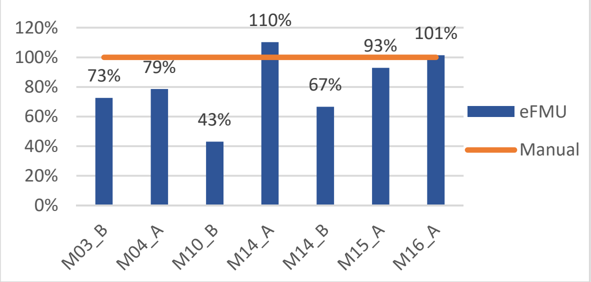


Figure 3. Run time measurements of eFMU production code with respect to manually coded solutions

The results of the performance benchmarks are shown in Figure 3. In 4 out of 6 examples (counting M14_A and M14_B as one), there is at least one eFMI tool chain setup that outperforms the manual implementation.

In case of *M10_B*, the manually derived solution of the inverse slider crank mechanism did not show a stable behavior unless two of the state variables were computed in double precision, while the auto-generated solution worked fine in single precision due to a more appropriate state selection.

The eFMU derived from the component-oriented rectifier model (*M14_A*) did not lead to the desired most compact and efficient formulation of the problem but gave very good results after reformulating the problem in the Modelica code (*M14_B*).

The reduced order model (ROM) of a thermal heat transfer test case (*M16_A*) has been processed by the tool chain starting from a manual implementation of the matrix equation system in GALEC code. As discussed in Agosta, et al. (2019), rigid scalarization, as applied by today's Modelica compilers, leads to an undesired code expansion. Here is room for improvement of the GALEC code generating tools. However, as the results show, the GALEC language is expressive enough to formulate this type of problem in a way that can be handled by the production code generating tools in a highly efficient way (the manually written GALEC code leverages on multi-dimensional arithmetic to avoid scalarization of the two-dimensional maps of the test case).

3.4 Code Quality Assessment

For all the test cases in Section 3.1, the code quality of the generated C code has been assessed by a static code analysis tool to find runtime issues such as variable overflows, possible division by zero, array index out of bounds, etc. or prove their absence. Also, the compliance of the code with the MISRA C:2012 rules has been checked. A static analysis is sound, but not necessarily complete. Hence, checked errors and rule violations are never overlooked, but may yield false alarms. Manual inspections resolved many of the false alarms so that in the 402 Production Code containers finally only 1% definite errors and 9% rule violations were detected. The main part of rule violations was detected in the implementations of built-in functions not being in the focus so far. It was assumed, that target-specific libraries realizing built-in functions will be used. Since then the tool prototypes have been further improved aiming for a full coverage of the MISRA C:2012 rules relevant for generated code.

3.5 Gain in Productivity

Aiming to put the time saving of an automated tool chain into perspective of the overall development effort of an embedded function, the working hours for modeling, implementation in C and validation of the results on the ECU have been counted for both the eFMI workflow and the manual development for the six benchmark examples.

The comparison of the results shows that in those cases based on a component-oriented modeling (*M03_B*, *M04_A* and *M10_A*) with a high level of reuse, the eFMI workflow took about 10 times less effort. For *M15_A* and *M16_A* the models have been implemented from scratch in Modelica based on a known state space formulation, but still gave a gain by a factor of 2.0 and 1.2 respectively. This stresses the high business value of eFMI for embedded software development especially for advanced physics-based control functions.

4 eFMI Applications

4.1 EMPHYSIS Demonstrators

The developed demonstrators, presented to the ITEA review board on Feb. 10, 2021 and summarized in the final demonstrator report of EMPHYSIS (2021-08), illustrate the application of the eFMI tool chain in concrete and realistic usage scenarios. These cover the domains vehicle dynamics, powertrain (internal combustion engine, battery electric vehicle, hybrid electric vehicle) and thermal systems and they are applied to advanced non-linear controllers, model-based diagnosis, virtual sensors and HiL simulation.

Renault demonstrated in two applications how a neural network trained by a high-fidelity model can be integrated as very accurate approximation into the embedded software running on a car by using the eFMI tool chain.

DLR-SR realized an advanced vertical dynamics controller and observer for semi-active damping (see Figure 4) using an inverse non-linear model and a non-linear Kalman filter running on a small series ECU in real driving tests. Never before for the institute, C code derived from a Modelica model has been directly integrated into the application software as in this case from the generated eFMI Production Code container.

GIPSA-lab demonstrated how eFMI can be utilized to derive a parametric Non-linear Model Predictive Controller (pNMPC) and deploy its production code to a dSPACE MicroAutoBox II ECU. The developed controller uses a neural network model to predict the future behavior of the car like the response of chassis and wheel to a given road profile and suspension parameter; this prediction is used for suspension control.

Volvo Cars demonstrated the development of an embedded virtual sensor for electric machine control based on a Modelica transmission model. The virtual sensor provides vehicle state estimation used to mitigate, e.g., backlash in the electric driveline, and thereby increase the overall performance of the whole electric driveline. The transmission model physics comprise non-linearities and discrete events for handling brake-torques at low speeds, resulting in a stiff discontinuous system with mixed equations that has been successfully

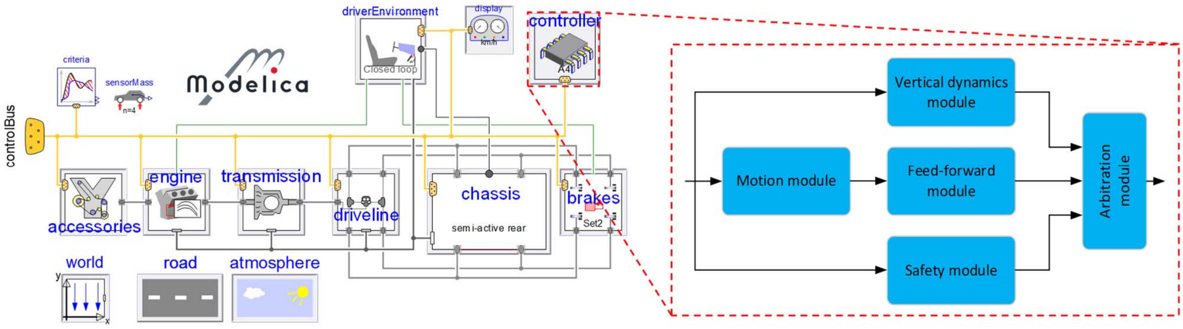


Figure 4. High fidelity vehicle model and advanced non-linear semi-active damping controller.

transformed by Modelica-tooling to a real-time suited GALEC solution.

Dassault Systèmes demonstrated the generation and validation of an AUTOSAR Adaptive Platform component starting from a Modelica model via a seamless, eFMI-based tool chain, for an advanced emergency braking (AEBS) controller. The AEBS controller is modeled in a classic block-diagram style with embedded physics. The blocks include enabled subsystems and signal locks, whereas the side effects of such are correctly handled using Modelica state machines.

4.2 OEM Advisory Board Feedback: Dual-clutch Transmission Demonstrator

The EMPHYSIS project has been accompanied by the so-called OEM Advisory Board with representatives from European and Japanese automotive OEMs. During half-day workshops, intermediate results of the project have been presented and discussed. An OEM Advisory Board usage scenario – a virtual sensor for a dual-clutch transmission – has been defined and a corresponding demonstrator implemented and evaluated in close collaboration between EMPHYSIS partners and the experts at Mercedes-Benz AG that provided the plant model of the dual-clutch transmission.

The objective of the virtual sensor is to use the physical model of a dual-clutch transmission to estimate the torque of clutches during shifting to avoid, for example, clutch over-burn and improve the driving-comfort during transmission shifting. The used transmission model had been derived from an existing high-fidelity system simulation model used in the product development of Mercedes-Benz AG; the most challenging system properties for a real-time application are therefore preserved. This includes the stiff dynamics of a hydraulic piston being tightly coupled with the discontinuous mode switching behavior of the clutches due to Coulomb friction, yielding a mixed equation system with undesired jittering even at a very small step-size of 0.1 ms with Explicit Euler.

By using a Rosenbrock method of order 1 (Hairer 1996) this problem could be drastically relaxed towards a jitter free behavior at a fixed step-size of 0.1 ms and robust but slightly jittering at a step-size of 10 ms. Compared to Explicit Euler, the Rosenbrock method therefore enables a factor 100 lower sampling rate, enabling the usage of the dual-clutch transmission model for embedded[1] real-time simulation for the very first time.

According to Mercedes-Benz AG, this result was considered a big progress towards using eFMI to derive very accurate plant models for SiL, HiL and embedded observer applications in a seamless fashion from a high-fidelity system simulation. It was confirmed that there is currently no better automated solution available for this task. As of today, a dedicated real-time model must be derived and individually fitted for each application causing significant repeated effort.

The eFMI container architecture with its built-in traceability and safety mechanisms has been praised by all members of the OEM Advisory Board as making eFMI a promising candidate to become the prescribed format for embedded software deliverables in OEM supplier collaborations. Especially for advanced functions like observers, the proposed eFMI workflow was considered as game changing technology to revolutionize the embedded software development.

5 Future Work

From the very beginning of the EMPHYSIS project (Lenord, 2019), also an Equation Code model representation has been investigated as an optional first intermediate target representation for acausal equation-based modeling tools. The motivation is that on the one hand Algorithm Code model representations can be generated from such a standardized, universal – but still simple – equation language, whereas on the other hand further equation analysis tooling could be integrated to

[1] The model has been used for real-time simulation by Mercedes-Benz AG before, but not for developing a software solution that can be deployed on the embedded device; this became possible only with eFMI.

refine equations or derive system characteristics of interest in the embedded domain like fault-behavior/safety, numeric stability etc.

To that end, a collaborative working group between EMPHYSIS partners and the Modelica Language working group has been formed, with the objective to define a proposal of a standardized *Flat Modelica* language as basis for a more restricted equation code language. A subset of Modelica keywords and a modified grammar have been proposed (Modelica Association 2021-06).

By implementing a prototypical *Flat Modelica* parser and pretty-printer for a non-Modelica tool within a few person months, it was demonstrated that other, already existing equation-based modeling tools (with their existing model representations, analyses capabilities and code generation back ends) can be integrated into the acausal modeling process with comparatively small effort. This early prototype tooling has been applied to two Bosch use cases: (1) inversion of a plant model of a drivetrain and (2) structural analysis of a thermal system to evaluate the detectability of system faults (EMPHYSIS 2021-08).

The work on a standardized *Flat Modelica* language and Equation Code model representation is planned to be continued and incorporated into a later version of the eFMI Standard.

6 Conclusions

This paper presented eFMI, a new workflow and open standard for the automatic generation of embedded software from physical models. The novelty of eFMI is its tooling-open, standardized exchange format by means of a container architecture with various standardized, traceable model representations for behavioral reference results, abstract algorithmic solutions, actual production codes and target-specific binary codes, bridging the gap between physics-modeling and embedded software.

A broad set of test cases, including technically challenging models, has been used to rigorously test and crosscheck the developed prototypical eFMI tools and their interoperability in the eFMI workflow. Together with performance benchmarks and code quality assessments, a high level of maturity has been testified.

The eFMI container architecture, with its various model representations, has been successfully applied to industrial usage scenarios. Automotive OEMs and Tier 1 suppliers confirmed the benefits of the proposed workflow over the state-of-the-art development processes in terms of repeatability, traceability and overall gain in productivity for embedded software development.

The work of EMPHYSIS and the eFMI Standard is continued in a new Modelica Association Project eFMI (MAP eFMI), successfully founded by core partners of the EMPHYSIS project. The work to further develop the eFMI specification towards a first official release according to established Modelica Association processes has already started. Companies and other organizations are encouraged to join MAP eFMI, leverage on the already developed tooling and foster the eFMI ecosystem.

Acknowledgements

This work is the result of the European ITEA3 Call2 EMPHYSIS (project number 15016). The work was funded by the German *Federal Ministry of Education and Research* (BMBF, grant numbers 01|S17023(A-H)), *Sweden's Innovation Agency* (VINNOVA, project number: 2017-05121), the French *Directorate General for Enterprise* and the Belgian agency *Flanders Innovation & Entrepreneurship*. The authors are responsible for the content of this publication.

The authors would like to thank the members of the EMPHYSIS OEM Advisory Board: BMW, Mercedes-Benz AG, Mazda, Volvo Trucks, JSAE for their use case, feedback and directions and the numerous persons that have worked in the EMPHYSIS project on the development of the eFMI Standard and/or evaluated it with tool prototypes, benchmarks and test cases as listed in appendix A of the eFMI specification.

References

Agosta, Giovanni, Emanuele Baldino, Francesco Casella, Stefano Cherubin, Alberto Leva and Federico Terraneo (2019). “Towards a High-Performance Modelica Compiler.” In: *Proceedings of the 13th International Modelica Conference*. Modelica Association, pp. 313–320. DOI: 10.3384/ecp19157313.

Armugham, Siva Sankar, Christian Bertsch, Karthikeyan Ramachandran, Oliver Lenord and Kai Werther (2021). “eFMI (FMI for embedded systems) in AUTOSAR for Next Generation Automotive Software Development”. In: *Symposium on International Automotive Technology 2021*. SAE International. Accepted March 31, 2021.

AUTOSAR Consortium (2021). *AUTOSAR (AUTomotive Open System ARchitecture)*. URL: http://www.autosar.org/.

Bertsch, Christian, Jonathan Neudorfer, Elmar Ahle, Siva Sankar Arumugham, Karthikeyan Ramachandran and Andreas Thuy (2015). “FMI for Physical Models on Automotive Embedded Targets”. In: *Proceedings of the 11th International Modelica Conference*. Modelica Association, pp. 43–50. DOI: 10.3384/ecp1511843.

EMPHYSIS (2021). *EMPHYSIS (Embedded systems with physical models in the production code software)*. URL: https://emphysis.github.io/.

EMPHYSIS (2021-07). *Functional Mock-Up Interface for embedded systems (eFMI)*. Version 1.0.0-alpha.4, Tech. rep.: EMPHYSIS Consortium. URL: https://emphysis.github.io/downloads.

EMPHYSIS (2021-08). *eFMI for Physics-Based ECU Controllers*. Tech. rep. D7.9, EMPHYSIS project deliverables: EMPHYSIS Consortium. URL: https://emphysis.github.io/downloads.

Englert, Tobias, Andreas Völz, Felix Mesmer, Sönke Rhein and Knut Graichen (2019). “A Software Framework for Embedded Nonlinear Model Predictive Control Using a Gradient-Based Augmented Lagrangian Approach

(GRAMPC)". In: *Optimization and Engineering* 20 (3), pp. 769–809. Springer. DOI: 10.1007/s11081-018-9417-2.

Hairer, Ernst and Gerhard Wanner (1996). *Solving Ordinary Differential Equations II*. 2nd ed. Springer. ISBN: 978-3-540-60452-5.

IEEE (2019-07). *IEEE Standard for Floating-Point Arithmetic*. Institute of Electrical and Electronics Engineers. ISBN: 978-1-5044-5924-2.

Intel Corporation (2021-06). *Intel® 64 and IA-32 Architectures Software Developer's Manual – Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4*. Tech. rep.: Intel Corporation. Order Number: 325462-075US.

ISO/IEC (2018-06). *ISO/IEC 9899:2018 — Information technology — Programming languages — C*. International Organization for Standardization.

Lenord, Oliver (2019). "Standardizing eFMI for Embedded Systems with Physical Models in the Production Code Software". Presented at: *Jubilee Symposium: Future Directions of System Modeling and Simulation*. Medicon Village, Lund, Sweden, September 30, 2019. URL: https://modelica.github.io/Symposium2019/.

MISRA (2013-03). *MISRA C:2012 – Guidelines for the use of the C language in critical systems*. MISRA Consortium Limited. ISBN: 978-1-906400-10-1.

Modelica Association (2021-02). *Modelica® – A Unified Object-Oriented Language for Systems Modeling – Language Specification – Version 3.5*. Tech. rep.: Modelica Association. URL: https://modelica.org/documents/MLS.pdf.

Modelica Association (2021-04). *Functional Mock-up Interface for Model Exchange and Co-Simulation*. Tech. rep.: Modelica Association. URL: https://fmi-standard.org/downloads/.

Modelica Association (2021-06). "Modelica Language Change Proposal 31 (MCP 31)". URL: https://github.com/modelica/ModelicaSpecification/tree/MCP/0031/RationaleMCP/0031.

Modelica Association (2021-07). "Official eFMI test cases for demonstrating and evaluating eFMI tooling". URL: https://github.com/modelica/efmi-testcases.

Neudorfer, Jonathan, Siva Sankar Armugham, Mathews Peter, Naresh Mandipalli, Karthikeyan Ramachandran, Christian Bertsch and Isidro Corral (2017). "FMI for Physics-Based Models on AUTOSAR Platforms". In: *Symposium on International Automotive Technology 2017*. SAE International. DOI: 10.4271/2017-26-0358.

Rüger, Johannes-Joerg, Alexander Wernet, Hasan-Ferit Kececi and Thomas Thiel (2014). "MDG1: The New, Scalable, and Powerful ECU Platform from Bosch". In: *Proceedings of the FISITA 2012 World Automotive Congress*. Vol. 6. Vehicle Electronics. Springer.

Wagner, Alexandre, Thomas Bleile, Slobodanka Lux and Christian Fleck (2009). "Method for real time capability simulation of an air system model of an internal combustion engine". Patent: United States US8321172B2, filed November 19, 2009.

Zimmermann, Michael, Thomas Bleile, Friedrun Heiber and Alexander Henle (2015). "Komplexitätsbeherrschung von Motorsteuerungs-Funktionalitäten". In: *MTZ - Motortechnische Zeitschrift* 76 (1), pp. 60–64. Springer. DOI: 10.1007/s35146-014-2003-z.

Modia – Equation Based Modeling and Domain Specific Algorithms

Hilding Elmqvist[1] Martin Otter[2] Andrea Neumayr[2] Gerhard Hippmann[2]

[1]Mogram AB, Sweden
[2]DLR Institute of System Dynamics and Control, Germany

Abstract

A new design of the Modia experimental modeling language based on Julia is presented. It has simple yet powerful syntax and semantics. A unified means of describing the fundamental semantics, which is similar to Modelica, is outlined. Furthermore, it is shown how domain specific algorithms can be combined with equation based modeling. It is demonstrated for multibody systems and enables more efficient translation since much repetitive analysis and transformations are avoided and faster simulation. The drive train of a robot model was automatically translated from Modelica to Modia. Modern simulation algorithms from the Julia community allow working with automatic differentiation and uncertainties.

Keywords: Modelica, Julia, Modia, Uncertainties, Multibody

1 Introduction

Modelica[1] (Modelica Association 2021), the equation and object oriented modeling language makes it easy to model large industrial systems with millions of equations. However, the transformation of the equations to executable form does not scale well since the current transformation technology is based on flattening of the hierarchical model structure. This means that certain structural and symbolic algorithms have to make repetitive work since the inherent structure is lost in the flattening process.

Furthermore, in the design of Modelica, several compromises needed to be made, for example, for multibody systems to handle spanning trees, closed kinematic loops, planar loops, over-determinism with quaternions, and reversing and zero flows for fluid systems. But even with these compromises, non-optimal performance of model transformation and simulation speed is achieved. On the other hand, the multibody community has designed methods for efficient simulation based on the manual conversion of the basic equations of motion and constraints to efficient algorithmic code, see e.g. (Arnold 2016).

Modelica also has the restriction that the number of equations and the number of states must be constant and that array dimensions of variables, even parameters, must be constant. There is a need for varying structure modeling to enable robot gripping, satellite docking, turning off parts of a fluid system, etc.

In this paper, we propose a hybrid solution: combining equation and object-oriented modeling with specialized algorithmic treatment of certain domains such as multibody and fluid systems. This approach relies on the powerful and fast programming language Julia[2] (Bezanson et al. 2017). The Julia package Modia[3] provides a modeling language that is based on hierarchical collections of name/value pairs. A unifying semantics has been defined for hierarchical modifiers à la Modelica and inheritance. Certain model instances are recognized as algorithmic models, i.e. calls to Julia functions are sorted among the solved equations. This technique also opens up for closed coupling to FEM and CFD models. Using a general-purpose algorithmic language instead of Modelica algorithms and functions enables use of more advanced data structures such as trees, dictionaries, etc.

It is important to have a stable model standard enabling encoding know-how available in books and articles in a formal language in order that these models can be stored and reused over a long time. The Modelica language was designed for this purpose. However, Modelica also needs to be enhanced due to various new needs within the modeling and simulation community. Modia was introduced to provide an experimental platform for extensions to the Modelica semantics and for experimentation of new transformation and simulation algorithms.

Since Modelica is a well-established modeling language there exist ten-thousands of models and it is very important to be able to reuse this huge model knowledge base. For that reason a translator between (so far for a subset of) Modelica and Modia is developed.

The presented modeling framework based on Julia has the advantage of using a modern infrastructure around the DifferentialEquations.jl package[4] (Rackauckas and Nie 2017b). For example, it enables using dual number representation for automatic differentiation and uncertainty information. Julia is also used for modeling in the ModelingToolKit (Ma et al. 2021) and experiments are made to use Julia instead of MetaModelica for the OpenModelica implementation (Tinnerholm et al. 2020).

[1] https://modelica.org/modelicalanguage.html

[2] https://julialang.org/

[3] https://github.com/ModiaSim/Modia.jl

[4] https://github.com/SciML/DifferentialEquations.jl

DOI
10.3384/ecp2118173

2 Modia Language

The Modia syntax and frontend has been redesigned since (Elmqvist, Henningsson, and Otter 2017). The new design is completely based on hierarchical collections of name/value pairs together with merging of such collections. This schema is used for models, variables, equations and hierarchical modifiers.

2.1 Variables and models

Variables are implicitly defined by their references in equations. A constructor `Var` allows defining variables with attributes:

```
name = Var(attribute=value, ...)
```

`Var` is a function taking name/value pairs, building and returning a corresponding dictionary.

```
Var(; kwargs...) = OrderedDict(kwargs)
```

Presently introduced attributes are: `value`, `min`, `max`, `init`, `start`, and the Booleans: `parameter`, `constant`, `input`, `output`, `potential` and `flow`. Example:

```
T = Var(parameter=true, value=0.2, min=0)
```

Some syntactically useful shortcuts using `Var` from **??** have been defined:

Listing 1. Media shortcuts.

```
Par(; kwargs...) =
    Var(; parameter=true, kwargs...)
input = Var(input=true)
output = Var(output=true)
potential = Var(potential=true)
flow = Var(flow=true)
```

If the value has references to other declared variables in the model, the expressions needs to be quoted that is enclosed in `:( )`. A parameter can also be defined by `name = literal-value`. `time` is a reserved name for the independent variable having unit s for seconds. The Julia package Unitful[5] provides a means for defining units and managing unit inference and checking. Definition of units is done with a string macro `u"..."` (see e.g., Listing 3). Units are given to inputs, states (`init`-attribute) and if the model equations contain systems of simultaneous equations, then approximate guess values, optionally with units, must be given as `start`-attribute to iteration variables.

A model (Listing 2) is also defined as a collection of name/value pairs with the constructor `Model` (similar to `Var`, but having a tag to enable better diagnostics).

Listing 2. Syntax of a Modia model.

```
name = Model(
  <variable-or-component-definition>,
  ...,
  equations = :[
    <equation1>
    <equation2>
    ...]
  )
```

The equations have Julia expressions in both left and right hand side of the equal sign. Note that the entire array of `equations` is quoted since enclosed in `:[ ]`. This enables later processing such as symbolically solving the equation since an AST (abstract syntax tree) is built-up instead of evaluating the expressions.

For example, in Modia a low pass filter can be defined as:

Listing 3. Modia model of a low pass filter.

```
LowPassFilter = Model(
  T = Par(value=0.2u"s", min=0u"s")
  u = input,
  y = output,
  x = Var(init=0),
  equations = :[
    T * der(x) + x = u
    y = x]
  )
```

This corresponds to the following Modelica model:

Listing 4. Modelica model of a low pass filter.

```
block LowPassFilter
  import Modelica.Blocks.Interfaces;
  parameter SIunits.Time T = 0.2;
  Interfaces.RealInput u;
  Interfaces.RealOutput y;
  Real x(start=0.0, fixed=true);
equation
  T * der(x) + x = u;
  y = x;
end LowPassFilter;
```

2.2 Connectors

Models which contain any flow variable, a variable having an attribute `flow = true`, are considered connectors. Connectors must have an equal number of flow and potential variables, variables that contain an attribute `potential = true`, and have matching array sizes. Connectors may not have any equations. An electrical connector with potential `v` (in Volt) and current `i` (in Ampere) is defined as:

```
Pin = Model(v = potential, i = flow)
```

2.3 Components

Components are declared by using a model name as a value in a name/value pair.

An electrical resistor with two Pins `p` and `n` can be described as follows:

Listing 5. Resistor model using Pins.

```
Resistor = Model(
  R = 1.0u"Ω",
  p = Pin,
  n = Pin,
  equations = :[
    0 = p.i + n.i
    v = p.v - n.v
    i = p.i
    R*i = v ]
  )
```

[5] https://github.com/PainterQubits/Unitful.jl

2.4 Merging

Models and variables are defined with hierarchical collections of name/value pairs. Setting and modifying parameters of components and attributes of variables are also naturally structured in the same way. A constructor `Map` is used for that. For example, modifying the parameter `T` of the `LowPassFilter` model defined in Listing 3 can be made by:

```
lowPassFilter = LowPassFilter |
    Map(T = Map(value=2u"s", min=1u"s"))
```

The achieved semantics is the same as for hierarchical modifiers in Modelica and results in:

```
lowPassFilter = Model(
   T = Var(parameter=true, value=2u"s",
           min=1u"s")
   ...)
```

The used merge operator `|` is an overloaded binary operator of bitwise or with recursive merge semantics[6]. In Listing 6 a sketch of the recursive merging function is given:

Listing 6. Merge operator `|` is an overloaded bitwise or.

```
function Base.:|(x, y)
  result = deepcopy(x)
  for (key, value) in y
    if typeof(value) <: AbstractDict &&
        key in keys(result)
      value = result[key] | value
    elseif key in keys(result) &&
        key == :equations
      equa = copy(result[key])
      push!(equa.args, value.args...)
      result[key] = equa
    end
    result[key] = value
  end
  return result
end
```

Merging of `equations` is handled specially by concatenating the equations vectors. More details, for example, about redeclaring and deleting names are given in the Modia tutorial[7].

2.5 Inheritance

Various physical components sometimes share common properties. One mechanism to handle this is to use inheritance. Modia makes a semantic unification by using *merging*.

Electrical components such as resistors, capacitors and inductors are categorized as oneports (Listing 7) that have two pins. Common properties are: constraint on currents at pins and definitions of voltage over the component and current through the component:

Listing 7. Oneport model for electrical components.

```
OnePort = Model(
  p = Pin,
  n = Pin,
  equations = :[
    0 = p.i + n.i
    v = p.v - n.v
    i = p.i ]
  )
```

Having such a `OnePort` definition (Listing 7) makes it convenient to define electrical component models by merging `OnePort` with specific parameter definitions with default values and equations:

Listing 8. Electrical components merged with `OnePort`.

```
Resistor = OnePort | Model( R = 1.0u"Ω",
  equations = :[ R*i = v ] )

Capacitor = OnePort | Model( C = 1.0u"F",
  v = Var(init=0.0u"V"),
  equations = :[ C*der(v) = i ] )

Inductor = OnePort | Model( L = 1.0u"H",
  i = Var(init=0.0u"A"),
  equations = :[ L*der(i) = v ] )

ConstantVoltage = OnePort | Model(
  V = 1.0u"V",
  equations = :[ v = V ] )
```

The resulting `Resistor` (Listing 8) defined by merging with `OnePort` is identical to the `Resistor` defined in Listing 5 due to the concatenation of the `equations` vector performed by the merge operator.

2.6 Connections

Connections are described as a special equation of the form:

```
connect( <connect-reference-1>,
         <connect-reference-2>, ... )
```

A 'connect-reference' has either the form 'connect instance name' or 'component instance name'.'connect instance name' with 'connect instance name' being either a connector instance, input or output variable.

For connectors, all the corresponding potentials of the connectors in the same connect statement are set equal. The sum of all incoming corresponding flows to the model are set equal to the sum of the corresponding flows into sub-components, i.e. the same semantics as in Modelica.

Connected models

Having the electrical component models from Listing 8 enables defining a filter (Listing 9), with internal resistance `Ri` of the voltage source, by instantiating components, setting parameters and defining connections.

Listing 9. Filter model defined with electrical components.

```
Filter = Model(
  R  = Resistor  | Map(R=0.5u"Ω"),
  Ri = Resistor  | Map(R=0.1u"Ω"),
  C  = Capacitor | Map(C=2.0u"F"),
  V  = ConstantVoltage | Map(V=10.0u"V"),
  equations = :[
    connect(V.p , Ri.n)
    connect(Ri.p, R.p)
    connect(R.n , C.p)
    connect(C.n , V.n) ]
  )
```

3 Transformation of Modelica models to Modia

A recursive-descent parser for Modelica has been developed in Julia by Hilding Elmqvist (Otter, Elmqvist, et al. 2019). It builds

[6] Python has also recently introduced the operator `|` for merging.

[7] https://modiasim.github.io/Modia.jl/stable/tutorial

an AST which is converted to the new Modia syntax.

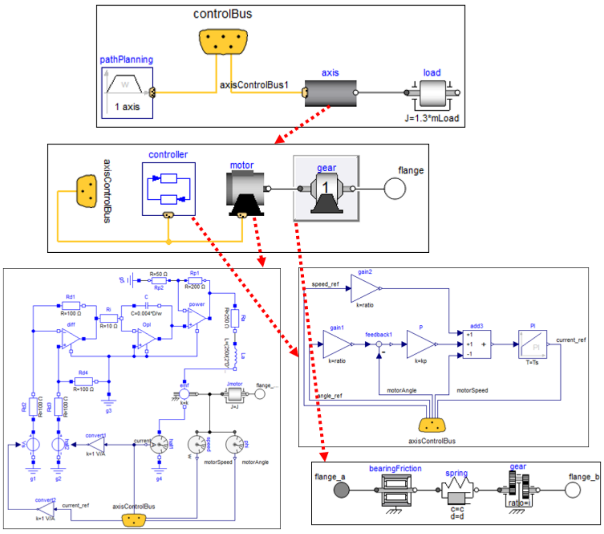


Figure 1. RobotR3.oneAxis drive line model.

This paper focuses on combining equation based modeling with algorithmic modeling of multibody systems. A good example of this combined need is robot modeling with the multibody part combined with rotational drive trains with inertias, gearboxes, springs, etc. and electrical motors with current amplifier electronics. All this is complemented with input/output blocks of the controllers. Such an example is available in the Modelica Standard Library[8] (Modelica Association 2020): Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3. The Modelica to Modia translator has been used to automatically translate the drive line model RobotR3.OneAxis in Figure 1 to Modia. The drive line consists of a path planning component, a P-PI controller, an electrical motor with current controller, a gearbox with friction, elasticity and damping and a rotational load inertia. By this translation, the know-how stored as Modelica models (55 models, 700 lines of Modia code) can be reused in an environment which enables more analyses than regular simulation as is demonstrated below.

4 Symbolic transformations

The Modia model `OneAxis` is instantiated with Julia macro `@instantiateModel(OneAxis)` that performs structural and symbolic transformations based on the algorithms sketched in (Otter and Elmqvist 2017), generates and compiles a function called `getDerivatives` for calculation of derivatives and returns a reference to the instantiated model where this function is stored. Transformation is currently performed to ODE form (Ordinary Differential Equations in state space form) where derivatives are explicitly solved for ($\mathbf{x}(t)$ is the state vector, $\mathbf{p}$ is a hierarchical dictionary of parameters and t is time):

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{p}, t), \quad \mathbf{x}_0 = \mathbf{x}(t_0) \tag{1}$$

Physical models lead often to linear systems of equations, as the `Filter` model in Listing 9. Media generates very compact code to built-up and solve linear systems of equations numerically during execution of the model, as shown for the `Filter` model in Listing 10.

Listing 10. Generated function for model `Filter`.

```
function getDerivatives(_der_x,_x,_m,_time)
  _p   = _m.evaluatedParameters
  _leq = nothing
  time = _time * upreferred(u"s")
  var"C.v" = _x[1] * u"V"
  var"V.v" = (_p[:V])[:V]
  var"C.p.v" = var"C.v" + -1var"V.v"
  begin
    local var"R.v", var"Ri.i", var"R.p.v"
    _leq = _m.linearEquations[1]
    _leq.mode = -3
    while leqIteration(_leq, <more arg.>)
      var"R.v" = _leq_mode.x[1]
      var"Ri.i"= var"R.v"/((_p[:R])[:R]*-1)
      var"R.p.v"= (_p[:Ri])[:R]*var"Ri.i"
      append!(_leq.residuals, stripUnit(
        var"R.v"-var"R.p.v"+var"C.p.v"))
    end
    _leq = nothing
  end
  var"der(C.v)" = -var"Ri.i" / (_p[:C])[:C]
  _der_x[1] = stripUnit(var"der(C.v)")
  if _m.storeResult
    addToResult!(_m,_der_x,time,var"R.v",
      var"R.p.v",var"Ri.i",var"C.p.v", var"
          V.v")
  end
  return nothing
end
```

Assume a nonlinear equation system

$$\mathbf{0} = \mathbf{g}(\mathbf{w}, \mathbf{v}) \tag{2}$$

has been identified with unknowns $\mathbf{w}$ and variables $\mathbf{v}$ that are known at this stage. If $\mathbf{g}$ is linear in $\mathbf{w}$, the tearing algorithm of (Otter and Elmqvist 2017) is applied to split this linear equation system in an explicitly solvable part $\mathbf{w}_2$ and an implicit part $\mathbf{w}_1$:

$$\begin{aligned} \mathbf{w}_2 &:= \mathbf{g}_1(\mathbf{w}_1, \mathbf{v}) & (3)\\ \mathbf{r} &:= \mathbf{g}_2(\mathbf{w}_1, \mathbf{w}_2, \mathbf{v}) \quad (= \mathbf{0}) & (4)\\ &= \mathbf{A}(\mathbf{w}_2, \mathbf{v})\mathbf{w}_1 - \mathbf{b}(\mathbf{w}_2, \mathbf{v}) & (5) \end{aligned}$$

Since it is known that (2) is linear in $\mathbf{w}$, it is possible to rearrange (conceptually) equations (3,4) into the form (5). This is, however, not actually done, because $\mathbf{A}$ has n^2 elements and then the size of the rearranged code would grow with $O(n^2)$. Instead, only code is generated to compute the residual $\mathbf{r}$, in order that the code size grows with $O(n)$:

```
while leqIteration(leq)
    $\mathbf{w}_1 :=$ leq.x
    $\mathbf{w}_2 := \mathbf{g}_1(\mathbf{w}_1, \mathbf{v})$
    leq.r $:= \mathbf{g}_2(\mathbf{w}_1, \mathbf{w}_2, \mathbf{v})$
end
```

Function *leqIteration* provides first leq.x $= \mathbf{0}$ and the while loop computes leq.r $:= -\mathbf{b}$. This vector is copied into an auxiliary vector inside the data structure *leq*. Afterwards, leq.x $= \mathbf{e}_i$ is set to the i-th unit vector and again the residual leq.r is computed. When varying i from 1 to n, all columns of $\mathbf{A}$ are computed. Afterwards the linear system $\mathbf{A}\mathbf{w}_1 = \mathbf{b}$ is solved and in a last

[8] https://github.com/modelica/ModelicaStandardLibrary

iteration the body of the while loop is again evaluated with the solution to get $\mathbf{w}_1 :=$ leq.x and compute $\mathbf{w}_2$.

During symbolic processing it is analyzed whether $\mathbf{A}$ is only a function of parameters $\mathbf{p}$, so does not change after initialization. In this case, the LU-decomposition of $\mathbf{A}$ is computed once during initialization and stored in the data structure *leq*. During simulation, only a (cheap) backwards solution is applied to compute the solution. If the residual equation has size one, a simple division is used, instead of calling a linear equation solver.

By default, the linear equation solver of Julia package *RecursiveFactorization.jl*[9] is used that implements the left-looking LU algorithm of (Toledo 1997). This solver is used up to a dimension of $n = 500$, because a benchmark shows a large speed-up with respect to the default linear solver based on OpenBLAS[10] that would otherwise be used.

The ODE and DAE integrators of Julia package *DifferentialEquations.jl*[11] (Rackauckas and Nie 2017a) are used with the generated `getDerivatives` function. If a DAE integrator is selected, the `getDerivatives` function is (automatically) called as needed by the interface of the DAE integrator.

Additionally, a powerful feature is included: If a DAE integrator is used and the size n of a linear equation system exceeds a particular limit (by default $n \geq 50$) and the unknowns $\mathbf{w}_1$ are a subset of the derivatives of the DAE states, then the following technique is used: During integration, the relevant DAE state derivatives are used as solutions leq.x of the linear equation system and the residuals leq.r are used as residuals for the DAE solver. The effect is that during integration no linear equation system is solved, but just the residuals leq.r of the linear equation system are computed *once* for every model evaluation. During *events* (including *initialization*), the linear equation system is constructed and solved and provides consistent initial conditions for the DAE solver. The big benefit is that simulation speed can increase tremendously, see the benchmarks in section 7.

5 Operations on Modia models

5.1 Simulation with parameter merging

The Modia model `OneAxis` (Listing 11) is instantiated, simulated and results plotted with the following commands:

Listing 11. Instantiate, simulate and plot results of model `OneAxis`.

```
using Modia
@usingModiaPlot
oneAxis = @instantiateModel(OneAxis)
simulate!(oneAxis, Tsit5(),
          stopTime=1.6u"s",
          merge=Map(load=Map(J=12.0)))
plot(oneAxis, [..])
```

Function `simulate!` performs one simulation of the instantiated model with a solver from the Julia package DifferentialEquations.jl[12] (Rackauckas and Nie 2017b). This package contains a large set of solvers. In Listing 11 the solver `Tsit5` is used. With various keyword arguments the simulation run can be defined. Especially, the stop time is set in the example to 1.6 *s*. If no unit is given, a unit of seconds is assumed. Furthermore, parameters and initial values can be provided by a hierarchical `Map` that is merged with the current values via the `merge` keyword. The simulation result is stored inside the instantiated model and is plotted with function call `plot`.

Hierarchical parameters and initial values can also be read from file, for example from a JSON file as shown in Listing 12.

Listing 12. JSON file for OneAxis parameterization.

```
{"axis": {
  "gear": {
    "ratio": 210.0,
    "c"    : 8.0,
    "d"    : 0.01,
    "Rv0"  : 0.5,
    "Rv1"  : 7.69e-4},

  "motor": {
    "J": 0.0013,
    "k": 1.616,
    "w": 5500.0,
    "D": 0.6,
    "w_max": 315.0,
    "i_max": 9.0},
 ...
}
```

The parameters and initial values read with function `readMap(..)` are stored in a hierarchical map that can be directly merged in to the model before simulation starts:

```
simulate!(oneAxis, Tsit5(),
  stopTime = 1.6u"s",
  merge=readMap("oneAxisParameters.json"))
```

Units can be defined using dictionaries (value, unit) as shown in Listing 13. These dictionaries are converted to Julia values with units before the merging is done.

Listing 13. JSON structure for parameterization with units.

```
{"axis": {
  "gear": {
  "ratio": 210.0,
    "c": {"value":8.0,"unit":"N/m"}
    ...
}
```

It is also possible to encode and decode such JSON parametrizations which contains Julia expressions for parameter propagation and calculations.

This offers new possibilities not available in the Modelica language: Since parameters and initial values are stored in a data structure, this data structure can be read from file or from a database system, then manipulated and finally simply merged into the model.

Typically, in Modelica parameter values are propagated and changes are performed via modifiers. For larger model hierarchies it is always hard to figure out which of the parameters to propagate and modify, because the set of parameters is too large. Sometimes records are used for the model parameterization, but then the corresponding models must be specially designed for these records, and the modeler ends up with a large set of different record types that cannot be conveniently utilized.

5.2 Simulation with different precisions

By default, a simulation is performed with 64 bit precision (Julia type `Float64`). However, the generated `getDerivatives` function does not depend on a particular type of the floating

[9] https://github.com/YingboMa/RecursiveFactorization.jl

[10] https://www.openblas.net/

[11] https://github.com/SciML/DifferentialEquations.jl

[12] https://github.com/SciML/DifferentialEquations.jl

point variables. Julia has a very elaborate type system and it is very easy to utilize different types when calling a function, provided a concrete type is not explicitly defined in the function signature. Note, the concrete type signature is given when calling the `getDerivatives` function from the solver, and then the function is specially compiled for this signature.

There is, however, one restriction: The used integrator must be prepared to use any type of floating point variables. This is the case for solvers that are natively defined in Julia, such as integrator `Tsit5`. DifferentialEquations.jl supports also external solvers that are typically implemented in `C` or `Fortran` for 64 bit precision only. These solvers cannot be used with other floating point types.

In Modia, the floating point precision is defined with the keyword `FloatType` of macro `@instantiateModel`. For example, the following definition simulates with 32 bit precision:

```
oneAxis = @instantiateModel(OneAxis,
                FloatType = Float32)
simulate!(oneAxis, ...)
```

Simulation can also be performed with higher precision:

- `FloatType = Double64` from package DoubleFloats.jl[13] uses two Float64 numbers and double word arithmetic to perform floating point operations with roughly 30 significant digits in an efficient way.

- `FloatType = BigFloat` is a Julia built-in floating point type wrapping the GNU Multiple Precision Arithmetic Library (GMP)[14] and the GNU MPFR Library[15] to support computations with any type of desired floating point precision. The drawback is that the computation might be slow.

5.3 Simulation with uncertainties

Julia package Measurements.jl[16] (Giordano 2016) provides calculations with uncertainties described by normal distributions using *linear error propagation theory*. This package allows to define uncertain variables with nominal value and standard deviation. For example $v = 2.0 \pm 0.2$ defines that variable v has a nominal value of 2.0 and a standard deviation of 0.2. In other words, with a probability of about 95 %, variable v is in the range $1.6 \leq v \leq 2.4$. The package overloads the Julia operators on floating point operations to perform propagation of uncertainties. This works also for functions, such as, solving linear equation systems. An example is given in Listing 14:

Listing 14. Uncertainty modeling with Measurements.jl.

```
using Measurements

v1 = 2.0 ± 0.2
v2 = 3.0 ± 0.3
v3 = v1 + v2          # = 5.0 ± 0.36
v4 = v3 - v1          # = 3.0 ± 0.3
v5 = v1*v2            # = 6.0 ± 0.85
```

In order to utilize this feature in Modia, the setting `FloatType = Measurement{Float64}` has to be used, defining that 64 bit floating point numbers with uncertainties are treated. Note, the uncertainty propagation again goes through all code, also through the integration algorithms. Therefore, this approach only works for solvers implemented in Julia. In Listing 15 this type of uncertainty modeling is applied on the `OneAxis` model, where uncertainties are defined for the load inertia `J` and the gear stiffness `c`:

Listing 15. Uncertainty modeling for OneAxis model.

```
using Modia, Measurements
@usingModiaPlot

OneAxis2 = OneAxis | Model(
  load = Map(J = 19.5 ± 4.0),
  axis = Map(c = 8.0 ± 0.8),
  angle_error = :(
    axis.axisControlBus.angle_ref-load.phi)
)

oneAxis2 = @instantiateModel(OneAxis2,
        FloatType=Measurement{Float64})
simulate!(oneAxis2, Tsit5(), stopTime=0.3)
plot(oneAxis2, "angle_error")
```

Function `plot` displays the nominal value as thicker line and the standard deviation as a transparent area around the nominal value, see Figure 2.

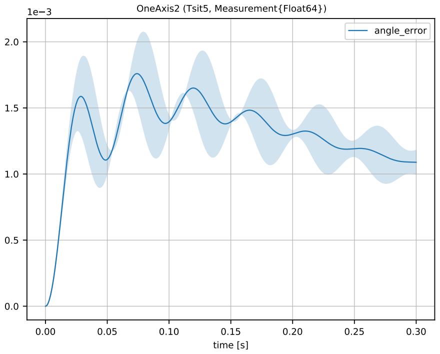


Figure 2. Control error of OneAxis model with nominal value (thick line) and standard deviation (transparent area).

Usage of the Measurements.jl package is attractive because, with very small effort, uncertainties of many variables can be defined and the propagated uncertainties of all variables computed in a reasonably efficient way. The drawbacks of this approach are that only normal distributions are supported and that uncertainty propagation is performed with linear theory, based on the analytic derivatives of all expressions. This means that larger parameter uncertainties will not be properly described by this approach and if the model contains discontinuous changes then the calculated standard deviations might be questionable.

5.4 Monte Carlo Simulation

Monte Carlo Simulation is a standard technique to evaluate a model with respect to uncertain parameters and initial values by randomly generating parameter and initial values with respect to given distributions and perform simulations for every randomly selected value set. The Julia package MonteCarloMeasurements.jl[17] (Carlson 2020) provides a variant via the nonlin-

[13] https://github.com/JuliaMath/DoubleFloats.jl

[14] https://gmplib.org/

[15] https://www.mpfr.org/

[16] https://github.com/JuliaPhysics/Measurements.jl

[17] https://github.com/baggepinnen/MonteCarloMeasurements.jl

ear propagation of arbitrary multivariate distributions by means of method overloading. This approach is attractive because the setup and usage is very simple, provided the underlying simulation environment is prepared to operate on any floating point type, as it is the case for Modia. A simple example of this package is given in Listing 16:

Listing 16. Example of MonteCarloMeasurements.jl.

```
using MonteCarloMeasurements, Distributions

uniform(vmin,vmax) = StaticParticles(5,
      Distributions.Uniform(vmin,vmax))

v1 = uniform(2.0, 3.0)  # v1.particles =
               # [2.7, 2.9, 2.3, 2.5, 2.1]
v2 = uniform(5.1, 8.3)
v3 = v1 + v2  # v3.particles =
         # [8.76, 10.24, 10.28, 9.2, 7.52]
```

Function `uniform` defines a uniform distribution between a minimum and maximum value. It generates five random values according to the given distribution and stores these five values internally in a vector. The five random values are just for illustration in this example. For realistic computations, typically several thousand random values are generated. The Julia package Distributions.jl[18] (Besançon et al. 2019) provides a large set of probability distributions and functions operating on them and can be used to generate a large variety of distributions for MonteCarloMeasurements.jl.

The package overloads all operations for floating point numbers by replacing for example the addition of two scalars by the addition of the two vectors, in which the randomly generated values are stored. Furthermore, the package is implemented to make effective use of SIMD[19] instructions available on modern processors. There is also support for computation on GPUs. To handle some corner cases, a few instructions in Modia had to be adapted to make Modia work with this package. In Listing 17, this type of Monte Carlo Simulation is applied on the `OneAxis` model, again, with uncertainties for the load inertia `J` and the gear stiffness `c` using 100 samples per distribution:

Listing 17. OneAxis model with MonteCarloMeasurements.jl.

```
using Modia,
using MonteCarloMeasurements, Distributions
@usingModiaPlot

uniform(vmin,vmax) = StaticParticles(100,
          Distributions.Uniform(vmin,vmax))

OneAxis3 = OneAxis | Model(
  load = Map(J = uniform(11.5, 27.5),
  axis = Map(c = uniform(6.6, 9.6),
  angle_error = :(
    axis.axisControlBus.angle_ref-load.phi)
)

oneAxis3 = @instantiateModel(OneAxis3,
  FloatType=StaticParticles{Float64,100})
simulate!(oneAxis3, Tsit5(), stopTime=0.3)
plot(oneAxis3, "angle_error")
```

Function `plot` displays the nominal value as thicker line and the particles of the corresponding variable as transparent, thin lines, see Figure 3.

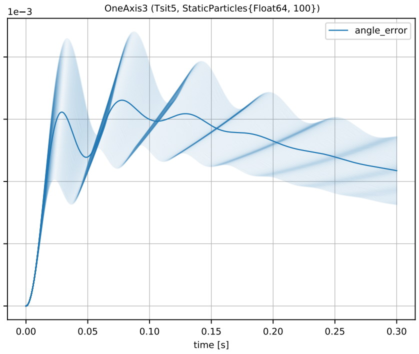


Figure 3. Control error of OneAxis model with mean value (thick line) and 100 particles (transparent, thin lines).

Usage of the MonteCarloMeasurements.jl package is attractive because, with very small effort, uncertainties of a reasonable amount of variables with *large uncertainties* can be defined for a large variety of probability distributions and the nonlinear propagation of these distributions is computed in an efficient way.

5.5 Linearization

An instantiated Modia model can be linearized:

```
using Modia
oneAxis = @instantiateModel(OneAxis)
(A0, x0) = linearize!(oneAxis)
(A1, x1) = linearize!(oneAxis, Tsit5(),
                  stopTime=1.0, analytic=true)
xNames   = get_xNames(oneAxis)
```

The first `linearize!` call initializes model `oneAxis`, computes the Jacobian of the state derivatives with respect to the states `x` numerically with a central finite difference approximation using Julia package *FiniteDiff.jl*[20], and returns the Jacobian as matrix `A0` together with the initial state vector `x0` computed during initialization. The nonlinear Modia model is hereby approximated at the initial state with the linear differential equation system $\Delta\dot{\mathbf{x}} = \mathbf{A}_0\Delta\mathbf{x}$, $\mathbf{x} \approx \mathbf{x}_0 + \Delta\mathbf{x}$. Linearization is performed with respect to the floating point type as defined by `FloatType`. If `FloatType = Measurement{Float64}`, the elements of matrix $\mathbf{A}$ are of this type, that is, contain uncertainties. Further processing is possible, especially with Julia package *ControlSystems.jl*[21] that also supports linear systems with uncertainties. In the near future, instantiation and linearization will be also supported with respect to top-level inputs and outputs of a Modia model.

The second `linearize!` call simulates the model with method `Tsit5` until the stop time, linearizes the system analytically using Julia package *ForwardDiff.jl*[22] (Revels, Lubin, and Papamarkou 2016) and returns the Jacobian as matrix `A1` together with the state vector `x1` at the stop time. Analytic differentiation may not always work. For example, an error is currently triggered if `FloatType =`

[18] https://github.com/JuliaStats/Distributions.jl

[19] Single Instruction, Multiple Data (= computers with multiple processing elements performing the same operation on multiple data points simultaneously).

[20] https://github.com/JuliaDiff/FiniteDiff.jl

[21] https://github.com/JuliaControl/ControlSystems.jl

[22] https://github.com/JuliaDiff/ForwardDiff.jl

Measurement{Float64} is used. Furthermore, analytic linearization takes some time, because the complete model is analytically differentiated, code generated and compiled.

Function call get_xNames(instantiatedModel) returns the names of the state vector $\mathbf{x}$ as a vector of strings. This allows to interpret further operations on the linearized system with respect to the nonlinear Modia model.

6 Modia with 3D models

6.1 Domain specific algorithms

Equation based modeling, for example with the Modelica language (Modelica Association 2021), maps a hierarchical model to a set of equations and transforms these equations appropriately. The drawback of this approach is that if a model contains, say, N instances of a body, the equations of the body are present N-times in the generated code. As a result, this approach does *not scale* for large models because the code size grows at least proportionally with the number of instances (and their number of equations) and therefore inherently limits the size of models that can be practically handled.

Traditional, domain-specific software, such as an electrical circuit simulator or a multibody program[23], have a completely different architecture: The equations of a component, say of a body, are available in a few variants and every variant is hardcoded in a function. If N bodies with the same variant are used, then the corresponding *function is called N times*, and the equations of the body are present only *once*. Therefore, the code size is independent from the size of the model. Additionally, special algorithms can be used, for example, to treat 3-dimensional rotations specially, handle over-determinism in planar kinematic loops, or use special sparse matrix methods, such as an $O(n)$ multibody algorithm. The drawback of this approach is that the introduction of new component types or the combination of submodels of different domains is orders of magnitude more difficult for a user to define than with equation based modeling.

In Modia, a new technique is utilized to combine the advantages of both approaches in a *generic way*, i.e., to combine equation-based modeling with domain-specific software. This approach is sketched and demonstrated in this section by combining equation based modeling with a multibody program[24].

In the simplest case, a multibody program for tree-structured systems basically has the following structure in pseudo-code notation, where $\mathbf{q}$ is the vector of generalized joint coordinates (for example angles of revolute joints) and $\mathbf{v}$ is the vector of generalized joint velocities:

$$\begin{aligned} &\texttt{mbs} = \texttt{readAndInit("fileName")} \\ &\dot{\mathbf{q}} = \mathbf{v} \\ &\dot{\mathbf{v}} = \mathbf{h}(\texttt{mbs}, \mathbf{q}, \mathbf{v}) \end{aligned} \tag{6}$$

The multibody system is defined on file. This file is read with function readAndInit(..) that returns an object reference mbs (so basically a pointer) of an internal data structure that allows fast evaluation of function $\mathbf{h}$ (6) which computes the derivative of $\mathbf{v}$. In the following it is shown how three basic issues are solved in Modia:

- Replacing the definition of the multibody system on file by a definition with the Modia language.
- Handle object references, such as mbs, in the Modia language.
- Handle state constraints, DAE index reduction and systems of equations that might occur when combining a multibody system with equation based models, for example when a drive train without gear elasticity (say, RobotR3.Utilities.AxisType2) is connected to a flange of a revolute joint.

6.2 Modia3D

Modia3D[25] (Neumayr and Otter 2018; Neumayr and Otter 2019a) is a Julia package that implements basically a multibody program, so targeted for solvers with adaptive step-size to compute results close to real physics, and combines this with the generic component-based design pattern of modern game engines. This allows a very flexible definition of 3D systems of any kind. Hereby, a coordinate system located in 3D is used as a primitive that has a *container with optional components* (such as geometry, visualization, dynamics, collision properties, light, camera, sound, etc.), see for example (Nystrom 2014)[26], Unity[27], Unreal Engine[28], three.js[29].

From a user's point of view, Modia3D provides a set of predefined model components (= constructor functions that generates dictionaries). The core component is Object3D that defines a coordinate system moving in 3D together with associated, optional features, see Figure 4. An Object3D is described relative to an optional parent Object3D by vector translation that defines the coordinates of the Object3D in its parent system and by vector rotation that defines three rotation angles to rotate the parent system into the Object3D system.

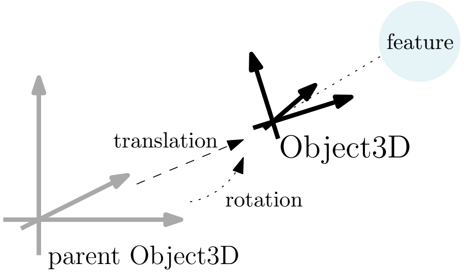


Figure 4. Object3D defined relative to its parent.

An example of a simple pendulum with damping in its joint is shown in Listing 18. The Object3D object that has feature Scene is the root of all other Object3Ds and defines a global inertial system. It is called world in the example. Object3D body defines a solid part that has a mass of 1 kg. On the body an Object 3D axle is defined that is translated 0.5 m along the negative x-axis of the body. Finally, a RevoluteWithFlange joint, that is a revolute joint with a flange, constrains the motion of axle with respect to world so that axle can only rotate around its z-axis.

[23] https://uwaterloo.ca/motion-research-group/multibody-system-dynamics-international-research-activities

[24] Modia uses an approach where *one ODE*-system is generated. The alternative of using co-simulation of *coupled ODE*-systems has inherent numerical issues and is not used.

[25] https://github.com/ModiaSim/Modia3D.jl

[26] http://gameprogrammingpatterns.com/component.html

[27] https://docs.unity3d.com/Manual/GameObjects.html

[28] https://docs.unrealengine.com/en-us/Engine/Components

[29] https://threejs.org/docs/index.html#api/core/Object3D

Listing 18. Simple pendulum with damping in its joint.

```
Pendulum = Model(
  # Multibody components
  world = Object3D(feature = Scene()),
  body  = Object3D(feature = Solid(
            massProperties =
              MassProperties(mass = 1.0))),
  axle  = Object3D(parent = :body,
            translation=[-0.5, 0.0, 0.0]),
  rev   = RevoluteWithFlange(axis=3,
            obj1=:world, obj2=:axis),

  # Equation based components
  damper  = Damper | Map(d=100.0),
  fixed   = Fixed,
  equations = :[
    connect(damper.flange_b, rev.flange),
    connect(damper.flange_a, fixed.flange)]
)

pendulum = @instantiateModel(
             buildModia3D(Pendulum),
             unitless=true)
simulate!(pendulum, stopTime=3.0)
plot(pendulum, "rev.phi")
```

The remaining elements of the Pendulum use predefined Models of a small Media library that corresponds to the Modelica.Mechanics.Rotational library. In particular a rotational 1D `Damper` is connected to a fixed point on one side and on the other side it is connected to the flange of the revolute joint.

Before calling `@instantiateModel(..)`, the special function `buildModia3D` must be called on the model to introduce a few equations to the model that depend on the used multibody components (details are given below). Additionally, option `unitless=true` has to be given temporarily, because units are not yet fully supported in Modia3D. The instantiated model can be simulated and variables plotted as before.

Feature `Visual` introduces objects for 3D animation by defining `shapes` like box, sphere, cylinder, 3D meshes on file in different formats (3ds, dxf, obj, stl), text, grids, or coordinate systems. A `visualMaterial` defines its visual properties. In Listing 19 an Object3D with a half transparent light blue cylinder with radius 0.01 and height 0.12 is created. For further details on visual objects see (Neumayr and Otter 2018, Section 2.2).

Listing 19. A visual Object3D with a light blue cylinder.

```
cylinder = Object3D(
  feature = Visual(
    shape = Cylinder(
      diameter = 0.01, length = 0.12),
    visualMaterial = VisualMaterial(
      color = "LightBlue",
      transparency = 0.5)))
)
```

Feature `Solid` defines solid bodies. Argument `shape` accepts primitive shapes and 3D meshes on file in obj-format. There are several ways for defining `massProperties`, including mass, center of mass, and inertia tensor of the solid: The default setting computes the mass properties from the density defined with the optional `solidMaterial` keyword and from the shape of `shape`. A solid object is considered in collision situations if keyword `collision` is set to `true`. Furthermore, it is possible to use keyword `collisionMaterial` to define properties of the collision behaviour, for example sliding friction coefficient and coefficient of restitution (Neumayr and Otter 2019b). In Listing 20 one link of a KUKA YouBot robot is defined as a solid 3D mesh with collision properties and its mass properties are computed from shape geometry and mass.

Listing 20. A solid Object3D that is allowed to collide and mass properties computed from shape geometry and mass.

```
body = Object3D(
  feature = Solid(
    shape = FileMesh(file =
      "YouBot/arm_joint_2.obj"),
    massProperties =
      MassPropertiesFromShapeAndMass(
        mass = 1.318),
    collision = true)
)
```

Feature `Scene` provides many options. A few are shown in Listing 21: With keyword `gravity` a uniform gravity field is defined pointing in negative z-direction. Only if `enableContactDetection` is set to true, collision handling is performed for all solid Object3Ds with enabled collision option.

Listing 21. World Object3D with scene defining a uniform gravity field pointing in negative z-direction and enabled contact detection.

```
world = Object3D(
  feature = Scene(
    gravity = UniformGravityField(
      g = 9.81, n = [0,0,-1]),
    enableContactDetection = true)
)
```

For modeling of freely moving bodies, without any kinematic constraint, Modia3D provides a `FreeMotion` joint with six degrees of freedom. It describes the orientation of Object3Ds by Tait-Byran (or Cardan) angles with rotation sequence x-y-z with respect to its parent Object3D, which is usually `world`. An advantage of this approach is that the state variables are illustrative for the user. Furthermore, in contrast to quaternions, Tait-Byran angles do not introduce nonlinear algebraic constraints and are directly defined in ODE form. If the main rotation is approximately around one axis (frequently given in technical applications) Tait-Byran angles behave nearly linear, so that integrators with adaptive step size selection can use larger steps compared to a description with quaternions.

However, a significant drawback of Tait-Byran angles is gimbal lock: When the second rotation about the local y-axis is

$$\alpha_2 = 90^\circ + n \cdot 180^\circ \quad n \in \mathbb{Z}, \tag{7}$$

x- and z-axes are parallel. In this configuration only the sum of the first and third angle is unique. As a consequence, the model equations become singular and simulation fails.

In Modia3D this situation is avoided by adaptive rotation sequence handling. For this, the second angle is monitored by a zero crossing function which stops time integration, if the absolute difference between its value and the gimbal lock condition (7) falls below a certain limit. Before restart, the `FreeMotion` joint is switched to the alternative rotation sequence x-z-y such that its relative orientation remains unchanged. Since x- and z-axes are nearly parallel, the new second Tait-Byran angle about the local z-axis obviously is far away from a gimbal lock configuration. After restart the same procedure is applied where the second angle about z is monitored and a switch back to rotation sequence x-y-z is triggered if the gimbal lock limit is reached again.

6.3 Modia3D implementation

In this section a sketch is given how the internal Modia3D structs and functions are interfaced with a Modia `Model`:

All Modia3D models, with exception of the joints, are defined in the following way:

Listing 22. Modia3D interface definition.

```
Object3D(;kwargs...) = Par(;
  _constructor =
     :(Modia3D.Composition.Object3D),
     kwargs...)

Visual(; kwargs...) = Par(;
  _constructor =
     :(Modia3D.Shapes.Visual), kwargs...)

Solid(; kwargs...) = Par(;
  _constructor =
    :(Modia3D.Shapes.Solid), kwargs...)
...
```

`Par` (see Listing 1) is a special `Var` provided by Modia and states that all keyword arguments are treated as parameters. The definition `Object3D(; kwargs...)=Par(...)` defines a *function* `Object3D` that has an arbitrary number of keyword arguments and calls Modia constructor `Par` with these keyword arguments, together with `_constructor = :(Modia3D.Composition.Object3D)`. Therefore, calling function `Object3D` returns a dictionary containing these keyword arguments together with the `_constructor` keyword argument.

For the code generation, Modia processes certain keywords of a parameter, for example, to access the parameter value in the generated Julia function. Other keywords, such as the Modia3D keywords, are ignored for the code generation. Before simulation starts, all parameters are *evaluated*. This means that the hierarchical dictionary of the parameter definitions are traversed recursively and parameter expressions and propagated parameters are evaluated. Furthermore, whenever a `_constructor` key is found, the corresponding constructor is called with the keyword arguments defined in that dictionary, the generated instance is compiled, and the value of the corresponding key (which was previously a dictionary) is replaced with a reference to the instance.

For example, `body = Object3D(..)` triggers a call of the constructor of the mutable struct `Object3D` in module `Modia3D.Composition` with the `feature` as keyword argument and the returned instance is used as *value* for key `body`. Some arguments of Modia3D components reference other Modia3D components, for example `axle = Object3D(parent = :body, ...)`. Since `:body` is a Julia `Symbol`, upper hierarchies of the hierarchical parameter dictionary are searched for a key corresponding to this symbol. Once found, the symbol used in `parent` is replaced by the *value* of key `body`, so by the reference to the body instance. After the parameter evaluation, the complete Modia3D data structure of this model is instantiated and available in the dictionary of *evaluated parameters*. Note, all this is *generic* and not specific to Modia3D.

In order that state constraints can be defined and index reduction performed, the interface to the Modia3D functionality is designed to define differential equations only on the Modia side. Since all Modia3D states are a subset of the generalized joint coordinates, part of the joint definition is done with the Modia language. For example, the definition of the `RevoluteWithFlange` joint is shown in Listing 23

Listing 23. Modia definition of a revolute joint with a flange.

```
RevoluteWithFlange(; obj1, obj2, axis=3,
    phi=Var(init=0.0), w=Var(init=0.0),
    canCollide=true) = Model(;
  _constructor = Par(value =
    :(Modia3D.Composition.Revolute),
    _jointType = :RevoluteWithFlange),

  obj1 = Par(value = obj1),
  obj2 = Par(value = obj2),
  axis = Par(value = axis),
  canCollide = Par(value = canCollide),
  flange = Flange, # defined in Rotational
  phi    = phi,
  w      = w,

  equations = :[
    phi = flange.phi
    w   = der(phi)]
)
```

This definition contains a `_constructor` variant, where only parts of the elements are parameters (defined with `Par`) and parts of the elements are Modia variables and equations. Only elements `_constructor`, `obj1`, `obj2`, `axis` are included in the parameter data structure. All other elements, especially `equations`, are processed in the usual way by Modia.

Function `buildModia3D(model)`, see Listing 18, recursively traverses `model`, so a hierarchical dictionary, and collects all information about the used joints (identified by `_constructor = Par(..., _jointType=xx)` together with the path name of this joint. Based on this information, the code from Listing 24 is merged to the model.

Listing 24. Code generated by `buildModia3D`(model).

```
model | Model(_id = rand(Int),
  equations = :[
    _mbs1 = initJoints!(_id,
       instantiatedModel,
       $ndofTotal, time)
    _mbs2 = setJointStates!(_mbs1,
       ($jointStates...))
    $jointForces = getJointForces!(_mbs2,
       _leq, ($jointAccelerations...))
    ]
)
```

The value of a Julia expression preceded by $ is inserted in the quoted expression, in this case, the ast of `equations` (see the result in Listing 25). Variable `_id` is a random number to provide a unique identification (details are given below). The `getDerivatives` function generated by Modia for the Pendulum example of Listing 18 is shown in Listing 25.

When the statement

```
_mbs1 = initJoints!(_p[:_id],_m,1,_time)
```

Listing 25. The `getDerivatives` function of the Pendulum example of Listing 18.

```
function getDerivatives(_der_x,_x,_m,_time)
  _p = _m.evaluatedParameters
  _leq = nothing
  var"rev.phi" = _x[1]
  var"rev.w"   = _x[2]
  _mbs1 = initJoints!(_p[:_id],_m,1,_time)
  _mbs2 = setJointStates!(_mbs1,
            var"rev.phi", var"rev.w")
  var"damper.tau" = (_p[:damper])[:d] *
                      var"rev.w"
  begin
    local var"der(rev.w)"
    _leq = _m.linearEquations[1]
    _leq.mode = -3
    while leqIteration(_leq, <more arg.>)
      var"der(rev.w)" = _leq.x[1]
      append!(_leq.residuals,
        getJointForces!(_mbs2, _leq,
          var"der(rev.w)") -
        SVector(-var"damper.tau"))
    end
    _leq = nothing
  end
  _der_x[1] = var"der(rev.phi)"
  _der_x[2] = var"der(rev.w)"
  ...
```

is called the first time, it traverses the evaluated parameters dictionary `_m.evaluatedParameters` until parameter `_id` with the provided value (`_p[:_id]`) is found. Afterwards, it inspects all `Object3D` instances in this subtree and checks that they are correctly defined, for example that exactly one of them has a `Scene` and that all Object3Ds are directly or indirectly connected to this object. Finally, an internal data structure is instantiated in which all needed information is stored, in particular references to the root Object3D `world`, and to the `Scene`. A reference to this data structure is stored in the dictionary `_m.userObjects[:_id]` by using `_id` as key. In all subsequent calls, this data structure is retrieved by accessing the dictionary. A reference to this data structure is returned as `_mbs1`. The statement
`_mbs2 = setJointStates!(_mbs1, ...)`
copies the joint states in to the internal multibody data structure. Function `getJointForces!(_mbs2,...)` computes the generalized joint forces from the generalized joint accelerations (here: `der(rev.w)`). Since the generalized joint accelerations are unknowns, this function call has to be inverted. This is performed by treating the function call as a residual equation of a linear equation system and the technique sketched in section 4 is used to solve this linear equation system with the while loop in Listing 25. If further equations are defined in the model as function of the unknown joint accelerations, for example inertias and ideal gear boxes connected to a flange of a joint, then these equations show also up in the body of the while-loop.

Note, the essential part of the multibody-related code - the three function calls - is independent of the size of the multibody-system. However, all states, derivatives of states and generalized forces of the joints are present in function `getDerivatives`, so the code size is linearly dependent on the degrees of freedom of the multibody system. But this code size is two to three orders of magnitude smaller, as a corresponding code of a Modelica multibody model.

6.4 Animation

Modia3D provides a generic interface to visualize simulation results with various 3D renderers:

- Both, the free community as well as the professional edition[30] of the *DLR Visualization* library[31] (Bellmann 2009; Hellerer, Bellmann, and Schlegel 2014) are supported that provide rendering during simulation and generation of videos in different formats at the end of the simulation.
- Another option is the automatic generation of a three.js JSON file at the end of the simulation. This file can be imported in the three.js editor[32] that allows flexible inspection of the animation and provides several ways for rendering the scene with different cameras and light options. Furthermore, the animation can be exported in the standard file format *glTF*[33] or its binary *glb* version for which many viewers are available. The initial configuration can also be exported in *obj*, *ply* or *stl* format.
- Moreover, an interesting feature of Microsoft Office 2019 (e.g. Word or PowerPoint) is the importing and rendering of these file formats. While for Office 2019 only a static rendering is possible, the latest Office 365 Subscription also supports playing the animation sequence.

6.5 Example: YouBot robot

The KUKA YouBot robot is a mobile robot with a 5 degree-of-freedom arm that was manufactured by KUKA in the years 2010-2016. This robot is an attractive benchmark because a lot of data, such as CAD drawings, visualization files, and solid data is freely available from the youbot-store[34]. The YouBot robot is modeled with Modia in a similar way as the `Pendulum` example, see Listing 18. In Figure 5, one Youbot is handing over a ball to another Youbot. The animation is stored in a JSON file, imported into three.js, exported in glb format and can be viewed with any glb viewer, such as the Windows 3D viewer included in Windows 10. The video of this example is available in the Modia3D tutorial[35]

7 Benchmarks

In order to evaluate the efficiency of Modia translations and simulations, the recursively defined benchmark model of figure 6 is used. It consists of a tree of solid wooden boxes and wooden spheres that are connected together with revolute joints in the form of a mobile. In every joint damping is present defined with `Damper` and `Fixed` components that are connected to the flange of the respective joint. With parameter `depth` the depth of the mobile model is defined and the Modia model[36] is recursively constructed. This model represents a reasonable mix of Modia language and of multibody components. The essential Modia code parts are shown in Listing 26.

[30] https://visualization.ltx.de/

[31] http://www.systemcontrolinnovationlab.de/the-dlr-visualization-library/

[32] https://threejs.org/editor/

[33] https://www.khronos.org/gltf

[34] http://www.youbot-store.com/wiki/index.php/YouBot_3D_Model

[35] https://modiasim.github.io/Modia3D.jl/resources/videos/-YouBotsGripping.mp4

[36] `Modia3D/test/Profiling/Mobile.jl`

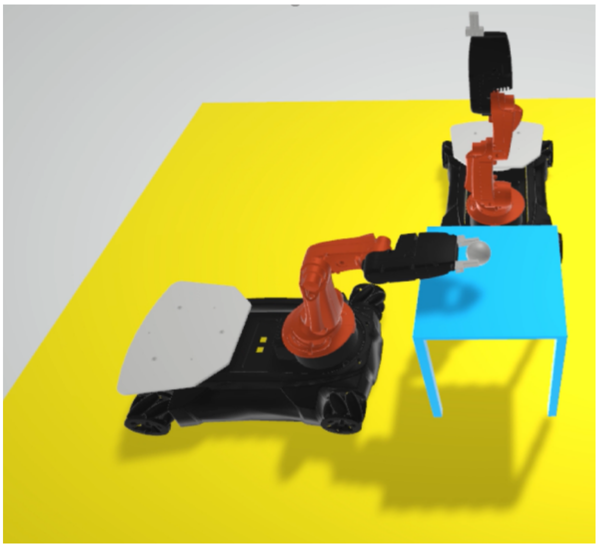


Figure 5. One YouBot handing over a ball to another Youbot (animation file in glb format, visualized with Windows 3D Viewer).

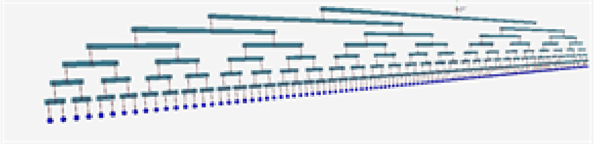


Figure 6. Recursively defined benchmark model `mobile` (here with depth=8).

Listing 26. Recursively defined benchmark model `mobile`.

```
function createMobile(depth)
  if depth == 1
    Model(
      rod    = Rod,
      sphere = Object3D(
                  parent=:(rod.frame0), ...)
  else
    Model(
      rod  = Rod,
      bar  = Bar | Map(L=barLength(depth)),
      sub1 = createMobile(depth-1),
      sub2 = createMobile(depth-1),
      rev0 = RevoluteWithDamping(
               obj1=:(rod.frame2),
               obj2=:(bar.frame0)),
      rev1 = RevoluteWithDamping(
               obj1=:(bar.frame1),
               obj2=:(sub1.rod.frame1)),
      rev2 = RevoluteWithDamping(
               obj1=:(bar.frame2),
               obj2=:(sub2.rod.frame1)))
  end
end
mobile = Model(
  world = Object3D(feature=Scene(..)),
  top   = createMobile(8),   # depth = 8
  rev0  = RevoluteWithDamping(
            obj1=:world,
            obj2=:(top.rod.frame1),
            phi_start=0.2))
```

Since a Modia model is basically a hierarchical dictionary, it can be constructed with the full power of the Julia programming language, and in particular with a recursive function. Essential properties of the benchmark model are summarized in Table 1. For various `depths`, simulations have been carried out for 5s with 500 communication points and a relative tolerance of 10^{-4}. All parameters are evaluated and animation and plotting is switched off. Equivalent Modelica models have also been constructed and simulations performed with OpenModelica[37] and two commercial Modelica tools. Comparing Modelica simulation tools with Modia can only be done very roughly, because the Modelica tools provide different timing information, or some timing information is not available and needs to be estimated with a stop watch. The timing given for Modia is the time to execute `@instantiate(..., logExecution=true)` in Table 2 and `simulate!(..)` in Table 3.

Table 1. Properties of the mobile benchmark.
#states are the number of ODE states.
#unknowns are the number of scalar unknowns of an equivalent Modelica model before alias elimination (up to 3 digits).
#solids are the number of solid boxes and spheres.
#joints are the number of *Revolute* joints (= number of *Damper* components).

depth	*#states*	*#unknowns*	*#solids*	*#joints*
1	2	603	4	1
2	8	1140	11	4
3	20	3590	25	10
4	44	7800	53	22
5	92	16300	109	46
6	188	33300	221	94
7	380	67000	445	190
8	764	135000	893	373
9	1532	271000	1790	766
10	3068	543000	3580	1534

Timings until simulation starts are given in Table 2 (column 2 gives absolute time and columns 3-5 timing factors relative to column 2).

The standard approach in Modia, *M-ODE*, provides the multibody equations in the form joint-forces = f1(joint-accelerations), so given the generalized accelerations in the joints, the generalized joint forces are computed, see subsection 6.3. This single equation can be combined with additional equations, for example an inertia can be attached directly to a revolute flange and then an additional equation is added that is a function of the joint acceleration. Also, a joint can be rheonomic, so the movement given. In all these cases, a linear system of equations is generated where the generalized joint accelerations, or angular accelerations in attached drive trains or generalized forces of rheonomic joints are the unknowns. So, this approach is general and allows to handle all cases that can appear in Modelica models.

The experimental approach *M-DAE*, provides instead the multibody equations in the form joint-accelerations = f2(joint-forces) and this function solves internally a linear system of equations over function f1(..). An error is raised if this function call appears in a system of equations. The effect is that, for example, it is no longer possible to attach an inertia of a drive

[37] https://www.openmodelica.org/

Table 2. Time-to-start-simulate including reading of model, symbolic transformations and
Modia: Generation of one Julia function, eval(..) of this function and executing this function twice.
Modelica tools: Generation of many C-Code functions, compilation, generation of executable and starting the exectuable.
M-DAE: Modia code: joint-accelerations = f2(joint-forces)
M-ODE: Modia code: joint-forces = f1(joint-accelerations)
tool1: The better of the two commercial Modelica tools.
tool2: OpenModelica 1.17.0.

depth	*M-DAE*	*M-ODE*	*tool 1*	*tool 2*
6	1.5s	× 3	× 15	× 120
7	4s	× 4.5	× 10	× 50
8	15s	× 3.5	× 4.3	× 90
9	50s	× 5	× 2.3	–
10	204s	–	–	–

train directly to a joint flange (a drive train must be attached with a compliant shaft). Also rheonomic joints cannot be used. The benefit of *M-DAE* is that the symbolic engine does not longer see an algebraic loop but only sortable statements, so the symbolic processing is faster and the generated code is smaller. Note, an alternative would be to use an O(n) algorithm, (Featherstone 1983) or (Brandl, Johanni, and Otter 1986), to compute the accelerations. The drawback would be, that the same restrictions as for *M-DAE* hold, e.g., it would not be possible to connect a 1D inertia to a joint flange.

As sketched at the end of section 4, when a DAE solver is used, all linear equation systems that exceed a given size are solved from the DAE solver *during integration*, provided all the unknowns of the linear equation system are a subset of the DAE state derivatives. The linear equation systems present for *M-ODE* and *M-DAE* fulfill the pre-requisites of this approach. The effect is that during integration no linear equation system is solved, but just the residuals of the linear equation system are computed.

For $depth \leq 5$, the Time-to-start-simulate of the Modelica tools is at least several seconds and is much longer as the actual simulation run, whereas the Modia simulation (both *M-ODE* and *M-DAE*) starts nearly immediately. For $depth \leq 8$, Time-to-start-simulate is below 15s for *M-DAE* and a factor of 4-120 larger for the Modelica tools.

For $depth > 10$, Time-to-start-simulate is no longer reasonable for *M-DAE* (and any other of the evaluated tools). The reason is that the generated Julia function becomes larger than a few thousand lines of code and then the quadratic increase of the Julia compilation time becomes dominant and limits the practical usage. For $depth = 10$, the compilation of the generated Julia code takes 174 seconds whereas the symbolic treatment of the model takes only 30 seconds. This barrier can be removed, because the generated Julia code can still be made more compact and also the technique of the Modelica tools can be used to split the computation in several functions.

The timings for the simulation runs are sketched in Table 3. As ODE integrator CVODE and as DAE integrator IDA from the Sundials suite (Hindmarsh et al. 2005) is used. Modia utilizes these solvers via Sundials.jl (Rackauckas and Nie 2017a). As can be seen, the simulation time of *M-DAE* is 1-2 orders of magnitude smaller than with the other solutions. The reason is that *M-ODE, tool 1* and *tool 2* solve a large, dense linear system of equations in every model evaluation, whereas *M-DAE* just computes the residuals of this equation system. The CVODE and IDA integrators calculate and factorize the system Jacobian for the benchmark simulation only about 10-20 times during one simulation run. This is just a small fraction of the linear equation systems that are solved inside every model evaluation of *M-ODE, tool 1* and *tool 2*.

Table 3. Simulation times for mobile benchmark.
M-ODE and the Modelica tools use Sundials CVODE and solve a linear equation system in the model.
M-DAE: Modia with Sundials.IDA() and residual algorithm.
M-ODE: Modia with Sundials.CVODE().
tool 1: The better of the two commercial Modelica tools.
tool 2: OpenModelica 1.17.0

depth	*M-DAE*	*M-ODE*	*tool 1*	*tool 2*
6	0.1s	× 30	× 30	× 240
7	0.3s	× 80	× 80	× 110
8	1.8s	× 100	× 180	× 120
9	10.5s	–	–	–
10	55s	–	–	–

8 Conclusion and Outlook

The paper outlines a path for utilization of available Modelica models in modern tools based on Julia, at the same time allowing integration of domain specialized models, such as multibody models, coded in Julia. In addition, the Modia language and new symbolic and numerical treatment of DAEs provide an experimental platform for developing new modeling capabilities. To make this path feasible, the translator from Modelica to Modia needs to be extended and be able to invoke domain specializations for multibody, fluid, media, etc. fully automatically.

It has been shown how simulation with uncertainties can be performed efficiently. A natural next step is to use these solver capabilities in the context of optimization and machine learning for surrogate models for speeding up simulations utilizing available packages from the Julia eco-system.

The Modia prototype handles benchmark models consisting of large multibody systems together with equation-based components much more efficiently as the examined Modelica tools - both for startup/compilation time as well as for simulation speed.

References

Arnold, Martin (2016). *DAE aspects of multibody systems*. Martin Luther University Halle-Wittenberg, Report No. 01. URL: http://sim.mathematik.uni-halle.de/reports/sources/2016/01-2016.pdf.

Bellmann, Tobias (2009). “Interactive Simulations and advanced Visualization with Modelica”. In: *Proceedings of the 7th International Modelica Conference*. LiU Electronic Press. DOI: 10.3384/ecp09430056.

Besançon, Mathieu et al. (2019). “Distributions.jl: Definition and Modeling of Probability Distributions in the JuliaStats Ecosystem”. In: *arXiv e-prints*. arXiv: 1907.08611 [stat.CO].

Bezanson, Jeff et al. (2017). “Julia: A fresh approach to numerical computing”. In: *SIAM review* 59.1, pp. 65–98. DOI: 10.1137/141000671.

Brandl, Helmut, Rainer Johanni, and Martin Otter (1986). “A Very Efficient Algorithm for the Simulation of Robots and Similar Multibody Systems without Inversion of the Mass Matrix”. In: *Proceedings of IFAC/IFIP/IMACS International Symposium on the Theory of Robots*. Elsevier. DOI: 10.1016/S1474-6670(17)59460-4.

Carlson, Fredrik Bagge (2020). “MonteCarloMeasurements.jl: Nonlinear Propagation of Arbitrary Multivariate Distributions by means of Method Overloading”. In: *arXiv e-prints*. arXiv: 2001.07625 `[cs.MS]`.

Elmqvist, Hilding, Toivo Henningsson, and Martin Otter (2017). “Innovations for Future Modelica”. In: *Proceedings of the 12th International Modelica Conference*. LiU Electronic Press. DOI: 10.3384/ecp17132693.

Featherstone, R. (1983). “The Calculation of Robot Dynamics Using Articulated-Body Inertias”. In: *International Journal of Robotics Research* 2.1, pp. 13–30. DOI: 10.1177/027836498300200102.

Giordano, Mosè (2016). “Uncertainty propagation with functionally correlated quantities”. In: *arXiv e-prints*. arXiv: 1610.08716 `[physics.data-an]`.

Hellerer, Matthias, Tobias Bellmann, and Florian Schlegel (2014). “The DLR Visualization Library – Recent development and applications”. In: *Proceedings of the 10th International Modelica Conference*. LiU Electronic Press. DOI: 10.3384/ECP14096899.

Hindmarsh, Alan C et al. (2005). “SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers”. In: *ACM Transactions on Mathematical Software (TOMS)* 31.3, pp. 363–396.

Ma, Yingbo et al. (2021). “ModelingToolkit: A Composable Graph Transformation System For Equation-Based Modeling”. In: *arXiv e-prints*. arXiv: 2103.05244 `[cs.MS]`.

Modelica Association (2020). *The Modelica Standard Library, Version 4.0.0*. URL: https://github.com/modelica/ModelicaStandardLibrary/.

Modelica Association (2021). *Modelica – A Unified Object-Oriented Language for Systems Modeling, Language Specification, Version 3.5*. URL: https://specification.modelica.org/maint/3.5/MLS.html.

Neumayr, Andrea and Martin Otter (2018). “Component-Based 3D Modeling of Dynamic Systems”. In: *Proceedings of the American Modelica Conference*. LiU Electronic Press. DOI: 10.3384/ECP18154175.

Neumayr, Andrea and Martin Otter (2019a). “Algorithms for Component-Based 3D Modeling”. In: *Proceedings of the 13th International Modelica Conference*. LiU Electronic Press. DOI: 10.3384/ecp19157383.

Neumayr, Andrea and Martin Otter (2019b). “Collision Handling with Elastic Response Calculation and Zero-Crossing Functions”. In: *Proceedings of the 9th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*. EOOLT’19. ACM, pp. 57–65. DOI: 10.1145/3365984.3365986.

Nystrom, Robert (2014). *Game Programming Patterns*. Genever Benning. ISBN: 978-0-9905829-0-8. URL: http://gameprogrammingpatterns.com/.

Otter, Martin and Hilding Elmqvist (2017). “Transformation of Differential Algebraic Array Equations to Index One Form”. In: *Proceedings of the 12th International Modelica Conference*. LiU Electronic Press. DOI: 10.3384/ecp17132565.

Otter, Martin, Hilding Elmqvist, et al. (2019). “Thermodynamic Property and Fluid Modeling with Modern Programming Language Construct”. In: *Proceedings of the 13th International Modelica Conference*. LiU Electronic Press. DOI: 10.3384/ecp19157589.

Rackauckas, Christopher and Qing Nie (2017a). “Differentialequations. jl–a performant and feature-rich ecosystem for solving differential equations in julia”. In: *Journal of Open Research Software* 5.1.

Rackauckas, Christopher and Qing Nie (2017b). “DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia”. In: *The Journal of Open Research Software* 5.1. DOI: 10.5334/jors.151.

Revels, Jarrett, Miles Lubin, and Theodore Papamarkou (2016). “Forward-Mode Automatic Differentiation in Julia”. In: *arXiv e-prints*. arXiv: 1607.07892 `[cs.MS]`.

Tinnerholm, John et al. (2020). “Towards an Open-Source Modelica Compiler in Julia”. In: *Proceedings of Asian Modelica Conference 2020*. LiU Electronic Press. DOI: 10.3384/ecp2020174143.

Toledo, Sivan (1997). “Locality of Reference in LU Decomposition with Partial Pivoting”. In: *SIAM Journal on Matrix Analysis and Applications* 18.4, pp. 1065–1081. DOI: 10.1137/S0895479896297744.

Modia and Julia for Grey Box Modeling

Frederic Bruder[1] Lars Mikelsons[1]

[1] Chair of Mechatronics, Augsburg University, Germany, {frederic.bruder,lars.mikelsons}@uni-a.de

Abstract

During the process of modelling an existing dynamic physical system, it may be hard to capture some of the phenomena exactly on the basis of only textbook-equations. With measurement data from the real system, approximators like artificial neural networks can help improve the models. However, simulation and machine learning are usually done in different software applications. A unified environment for modeling, simulation and optimization would be highly valuable. We here present a framework within the Julia programming language that encompasses tools for acausal modeling, automatic differentiation rsp. sensitivity analysis involving solvers for differential equations. We use it to build and evaluate an easily interpretable model based on both physics and data.

Keywords: Grey Box Modeling, Hybrid Modeling, Scientific Machine Learning, Modia, Julia

1 Introduction

1.1 Usefulness of Acausal Modeling

Equation-based acausal modeling like in Modelica has considerable benefits for the model author when compared to signal-based modeling.

It allows them to structure model equations and variables hierarchically, which greatly promotes later reuse of the components. It lets the author focus on model topology rather than signal flow and its direction within the model. Connect equations instead of assignments permit signal flow in both directions so that the compiler can decide what the direction will be.

Well-established Modelica Compilers automatically improve the numerical behaviour of the models, e.g. by reducing the size of nonlinear systems with the help of tearing or by reducing the index of Differential-Algebraic-Equations (DAE).

In short: acausal modeling is highly useful because it eases the job for authors of white box (i.e. mechanistic physics-based) models.

1.2 Grey Box Modeling

In the case of GBM ('Grey Box Modeling' or '... Model'), the author combines approximators like ANN (artificial neural networks) with trusted white box model equations. The goal of this technique is to transfer physical knowledge into a model that can be improved with the help of machine learning. Others referred to this field as 'Scientific Machine Learning' (Rackauckas, Ma, et al. 2020), 'Hybrid Physics Guided Machine Learning' (Rai and Sahu 2020) or simply 'Hybrid Modeling' (Willard et al. 2020). There are different motivations to do this:

- Known white box models may fail to describe the dynamics of an existing system appropriately. In this case the insertion of ANN into a model may help to reduce model error w.r.t. ground truth data collected the real physical system. Imagine a situation in which you first model a physical system. Take, for example, a simple model to predict the temperature in a lake as in (Karpatne et al. 2018). Although you stuck to well-known mechanistic equations, you see that the prediction error is too high. Due to the nature of the errors, you suspect a systematic error, not a stochastic one. To improve your model, you have a few options: You could try and improve your model by altering the equations you used. You would have to think about your model assumptions and whether or not they hold. This may, after possibly a lot of work, provide you with new physical insights. If you are more interested in fast results or if your system is just too complicated to fully grasp, you may consider using a machine learning-assisted approach that your initial model as a starting point and optimizes the 'flexible' parts to improve the predictions.

- White box models may require more computational resources than their target platform offers. Performance-critical parts may then be exchanged by ANN in order to create more efficient surrogate models. Thus, this technique enables model authors to trade model accuracy for better performance. An example of this application can be found in (Ma et al. 2021).

There are some complications with well-established Modelica compilers when it comes to GBM:

- Small modifications of existing Modelica models may require substantial refactoring. For example, you may have to declare a few new (abstract) components if you just want to replace a single equation in a model.

- It may be hard for a user to interpret or edit the structure of causalized models. This is problematic when it is not yet clear where exactly the ANN shall be inserted into the model. ANN that are defined as mathematical functions typically have an 'input layer' that is nonlinearly transformed to an 'output layer'. Said transformation may be arbitrarily complex. During the causalization process of the model equations, signal flow may be 'reversed' w.r.t. the intended direction so that the ANN is bound to end up in a root finding loop. This might lead to subpar numerical performance. To prevent this situation, the model author may wish to decide where exactly the ANN is included **after** the causalization process. It is, however, not very convenient to deal with C-Code output from a Modelica compiler.

- Another issue is the ANN training process. ANN are typically optimized w.r.t. their training/hyper parameters in languages that promote fast prototyping (e.g. Python, Julia, R, ...) with the help of tools like automatic differentiation. The latter kind of tools often requires that the function to be differentiated be formulated in the same language as the prototyping language.

It would be much more convenient to have modeling, simulation, automatic differentiation / sensitivity analysis and optimization algorithms in a single unified environment suited for rapid prototyping.

1.3 Julia

Julia (Bezanson et al. 2012) is one of such languages that promote fast prototyping. Julia has become popular in the scientific community because of the high level of performance it can reach without the need to use precompiled libraries created in a different language. The latter workflow is common e.g. in Python. On top of that, an ecosystem of packages useful for scientific calculations has gathered around it. Moreover, Julia offers metaprogramming capabilities that make it possible to create DSL (domain-specific languages).

1.4 Modia / TinyModia for GBM

Modia (Elmqvist and Otter 2017) is such a DSL that can be used within Julia. Its authors describe it as a testbed for new features for the Modelica language that 'shall be both simpler and more powerful than Modelica 3.3'.

Modia offers features highly valuable to GBM since it solves some of the issues listed in 1.2. In our present research, we are, however, using TinyModia (Elmqvist and Otter 2021) v.0.7.2 that had some features not yet present Modia.

- Models are represented as hierarchical `NamedTuples`. They can be modified with a mechanism called 'merging', their `Array` fields can even be manipulated directly. So, after a model has been declared, it can be modified equation-wise. This allows for a very convenient GBM workflow: you can declare a base model and then derive different versions of grey box models where different sets of equations are modified.

  Whereas in Modelica, one would typically have to define new (replaceable) component `Models` with a different set of equations and then derive simulation `Models` as a combination of those new components.

- During model 'instantiation', a Julia method (`getDerivatives!`) that calculates the state derivatives of the model is generated with the help of metaprogramming. Due to the open nature of TinyModia, the AST (abstract syntax tree) that, when evaluated, defines this new method, can be altered as well.

  Modelica Compilers, on the other hand, do not expose the AST of the simulation code to Julia. While they output source code of the simulation model, working with this representation would be less convenient: one would have to modify the output C-Code manually, without Julia's metaprogramming functionality.

- With a modifiable method `getDerivatives!`, the model author can take advantage of packages other than TinyModia to simulate the model. With this workflow, it even becomes possible to perform AD through solvers using the scientific packages available for Julia.

1.5 Structure of this Paper

This paper is structured as follows. In section 2, we describe a planar slip-based vehicle model which we modified to fit to our tests. Using this model as an example, we elaborate how models can be turned into GBM by replacing a set of model equation terms by trainable neural networks. In section 3, we proceed by detailling the time horizon-based training scheme that we used to train said neural networks. We then describe how we designed our tests with the modified and optimized vehicle model in section 4 and discuss the results. After that in 5, we describe the framework of tools we used to set up a GBM. In addition, we describe possible features for TinyModia/Modia that would have made our workflow a lot easier. Next, we name some of our future research perspectives involving the mentioned tool set and the Grey Box methodology. We then finish with a conclusion.

The main contribution of this paper is the detailed framework of tools that we use for Grey Box Modelling. We describe how to set up a model and enhance it with artificial neural networks. We sketch a training loop that is capable of optimizing such grey box models. We make suggestions for new features that would have made this process a lot more straight-forward.

2 GBM of a Single Track Vehicle

2.1 Original Model and our Modifications

We applied the ideas explained above to an NLSTM (nonlinear single track model). It makes use a well-known slip-based tire model as in (Pacejka 2005). The original model without suspension dynamics from (Velenis, Frazzoli, and Tsiotras 2009), on which our model is based, focusses on stability of cornering maneuvers, whereas we needed a more general purpose model. So we chose an alternative set of dynamic states $\boldsymbol{u}$ for our NLSTM:

$$\boldsymbol{u} = \begin{bmatrix} \dot{x} & \dot{y} & \psi & \dot{\psi} & x & y & \omega_F & \omega_R \end{bmatrix}^T \tag{1}$$

x and y denote the location of the center of mass of the vehicle. ψ measures the yaw angle / heading direction of the vehicle, ω_F and ω_R describe the rotational velocity of the front and rear wheel. Furthermore, we modified the original slip-based friction model to be less numerically stiff. For this, each occurrence of the term on the RHS (right-hand side) of

$$\texttt{hard\_pole}(x) = |x|^{-1} \tag{2}$$

was replaced by the RHS of

$$\texttt{smooth\_pole}(x) = \begin{Bmatrix} |x|^{-1} & \text{if } |x| > x_i \\ -\frac{1}{2}|x_i|^{-3}x^2 + \frac{3}{2}|x_i|^{-1} & \text{else} \end{Bmatrix} \tag{3}$$

with $x_i = 10^{-3}$. `smooth_pole`(x), as the name suggests, is a symmetric continuously differentiable function that replaces the actual ‘pole region’ of `hard_pole`(x) for $x \in [-x_i, x_i]$ by an inverted parabola tangent to the original function. This modification effectively prevents divisions by zero.

To have velocity-dependent friction in the model, a **linear** air drag force $\boldsymbol{f}_{drag}$ was applied to the center or mass of the vehicle:

$$\boldsymbol{f}_{drag}(\boldsymbol{x}) = w \cdot \begin{bmatrix} \dot{x} & \dot{y} \end{bmatrix}^T \tag{4}$$

Air drag parameter w was set to 30 Ns/m.

Apart from these three modifications, our RM (reference model) and the original model share the same equations and parameters. Note however, that x_i was set arbitrarily. In a real application, it may be helpful to set this parameter, introduced to enhance numerical performance, based on collected data.

The model inputs are the same as those of the original model:

$$\boldsymbol{i}(t) = \begin{bmatrix} \delta(t) & T_F(t) & T_R(t) \end{bmatrix}^T \tag{5}$$

where δ is the steering angle, T_F and T_R are the engine torques acting on the front rsp. the rear wheel.

2.2 Replaced Equations

In the original model, normal forces acting between the ground and the wheels are denoted by f_{Rz} (rear wheel) and f_{Fz} (front wheel). Based on the original model equations, the following equations hold:

$$f_{Rz} = \frac{mgA}{A - h\mu_{Rx} + l_R} \tag{6}$$

$$f_{Fz} = mg - f_{Rz} \tag{7}$$

$$A = h(\mu_{Fx}cos(\delta) - \mu_{Fy}sin(\delta)) + l_F \tag{8}$$

m, g, h, l_R and l_F are fixed model parameters. A is a convenience variable introduced for readability. The remaining symbols in Table 1 are model variables which can change during simulation.

Table 1. remaining model variables in Equations 8 to 7

Symbol(s)	*Meaning*
δ	steering angle
μ_{Fx}, μ_{Fy}	friction quantities (front wheel)
μ_{Rx}	friction quantity (rear wheel)

Equations 8 to 7 can be restructured as:

$$\begin{bmatrix} f_{Rz} \\ f_{Fz} \end{bmatrix} = \boldsymbol{f}_{Xz}(h, \mu_{Fx}, \delta, \mu_{Fy}, l_F, m, g, \mu_{Rx}, l_R) = \boldsymbol{f}_{Xz}(\boldsymbol{v}) \tag{9}$$

$\boldsymbol{v}$ is a convenience vector holding all the function arguments. Like this, $\boldsymbol{f}_{Xz}$ calculates both f_{Rz} and f_{Fz} from arguments $\boldsymbol{v}$.

In order to turn the original model into a flexible GBM, function $\boldsymbol{f}_{Xz}$ is replaced by an ANN named $\hat{\boldsymbol{f}}_{Xz}(\boldsymbol{\theta}, \boldsymbol{v})$. It depends on $\boldsymbol{\theta}$, a set of training parameters and $\boldsymbol{v}$. It was built from fully connected layers as depicted in Table 2. This ANN was defined with the help of `Flux.jl` (Michael Innes et al. 2018) and is initialized using random numbers drawn from a `glorot_uniform` distribution.

Table 2. layers of ANN $\hat{\boldsymbol{f}}_{Xz}$

layer	*# inputs*	*# outputs*	*activation*
input	9	10	Relu
hidden	10	10	Relu
output	10	2	Identity

Note that the ANN takes as many inputs as the pair of equations it replaces. This would not be neccessary, one could exclude the fixed parameters and make the input space 4-dimensional instead of 9-dimensional. The input space is kept as large as that of $\boldsymbol{f}_{Xz}$ so that the knowledge about which model variables change and which do not is not required.

3 Model Training

3.1 Loss Function

The method into which the ANN was inserted specifies the RHS of an ODE (ordinary differential equation). So, `getDerivatives!` calculates $\hat{\dot{\boldsymbol{u}}}(\hat{\boldsymbol{u}}(t), \boldsymbol{\theta}, t)$.

In order to train the parameters of the ANN, a loss function $l(\boldsymbol{\theta})$ needs to be formulated. For this test, we chose an MSE-based (mean squared error) function:

$$l_{MSE}(\boldsymbol{\theta}, C, S) = \frac{1}{|C| \cdot |S|} \sum_{c \in C} \sum_{s \in S} (\hat{u}_s(\boldsymbol{\theta}, t = c) - u_s(t = c))^2 \tag{10}$$

l_{MSE} compares a single solution $\hat{\boldsymbol{u}}(\boldsymbol{\theta},t)$ of the ODE to a ground-truth solution $\boldsymbol{u}(t)$.

Normally, when ground-truth data is collected using sensors from real physical systems, $\boldsymbol{u}(t)$ is only known at *specific points in time*. We call these *checkpoints* and C denotes the set of checkpoints relevant to l_{MSE}.

S, on the other hand, is a set of states relevant to l_{MSE}. Typically, ground truth measurements cannot capture all of the dynamic states that would be neccessary to fully specify the state of an arbitrary model of the real system. Our system state in Equation 1 has 8 entries. If you can only measure, say, the planar position of the vehicle, S would hold only 2 elements. $u_s(t)$ rsp. $\hat{u}_s(t)$ denote the trajectories of single elements of the system states in time.

3.2 Training Procedure

When we first tried to minimize l_{MSE} with the full set of available checkpoints, we would end up with with very bad local minima. The solution trajectory to the model ODE would look like an underfitting approximator: instead of following the checkpoint trajectory closely, it would resemble a smoothed cutting-corners version of the checkpoint trajectory. This is due to the formulation of our loss function: it compares the model ODE trajectory against all of the measurements at the checkpoint times at the same time and may reach local minima with the last few checkpoints, whereas earlier checkpoints are far missed.

However, this is not what we wanted. If early model predictions are very far off, we do not care about whether the model later succeeds in hitting checkpoints again. We would rather expect later checkpoints to be missed even further. Unless we are dealing with systems that evolve to a stable equilibrium.

To overcome this issue, a **growing horizon** training scheme has been implemented. It adds an outer loop to our training loop: we start with horizon $C_0 = [t_0, t_1, ..., t_{init}]$ and apply our inner training loop to optimize $\boldsymbol{\theta}$. The inner loop then keeps iterating until we hit a breaking condition involving distance metric d_{max}:

$$d_{max}(\boldsymbol{\theta}, C_n, S) = \max_{c \in C_n, s \in S} |\hat{u}_s(\boldsymbol{\theta}, t=c) - u_s(t=c)| \qquad (11)$$

If d_{max} falls below threshold d_t, we save our last training parameter set $\boldsymbol{\theta}^{0*}$, grow the training horizon to $C_1 = [t_0, t_1, ..., t_{init}, t_{init+1}]$ and then further optimize $\boldsymbol{\theta}^{0*}$ over that new horizon. We repeat this process until our inner loop has optimized $\boldsymbol{\theta}$ over the final horizon $C_{final} = C_{|C|-init-1}$. This training scheme heavily improved the quality of our model predictions on the training set.

One may wonder why we constructed d_{max}, an arbitrary distance metric, to be compared against a threshold instead of governing l_{MSE} instead. Both metrics evaluate to 0 for a perfect $\boldsymbol{\theta}$. However, $d_{max} < d_t$ is a very intuitive condition: none of the trajectories of the states in S may deviate further than d_t off the recorded time series data at the checkpoint times. If l_{MSE} was compared to a fixed threshold instead, the condition would become more and more permissive towards single outliers the larger the horizon grows.

4 Evaluation

4.1 Generation of Training Data

For lack of a real vehicle on which to collect data, we made use of the original vehicle model to generate training data for our GBM. Our data generation is limited to open-loop DLC (double lane changes) that the vehicle performs after its initial state has been set to a straight slip-free state:

$$\boldsymbol{u}_0 = 30 \cdot [1,0,0,0,0,0,r_F^{-1},r_R^{-1}]^T \qquad (12)$$

Figure 1 and Table 3 illustrate how model input δ(t) is varied over time. During all DLC maneuvers, model inputs T_R and T_F are kept constant over time. T_R is fixed at 0, whereas the value of T_F is a maneuver parameter.

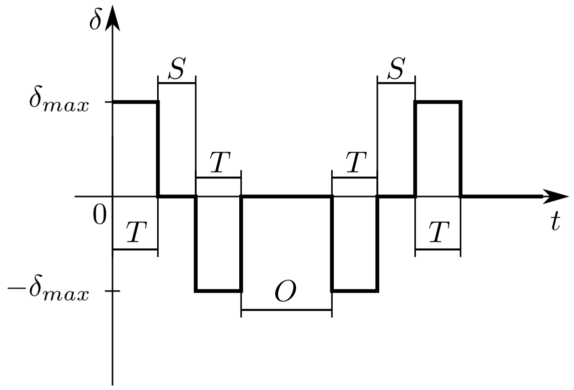


Figure 1. δ(t) during a DLC maneuver

Table 3. DLC parameters

Parameter	*Meaning*
δ_{max}	maximum absolute steering angle
T	**turn** time
S	lane **switch** time
O	time on the **oncoming** lane

Thus, a single DLC maneuver of ours is parametrized using a set of five parameters:

$$\boldsymbol{p}_{DLC} = [\delta_{max}, T, S, O, T_F] \qquad (13)$$

4.2 Results

In the following, the training dataset was limited **to only one** trajectory that was produced with the DLC parametrization shown in Equation 14.

$$\boldsymbol{p}_{DLC}^{train} = \boldsymbol{p}_{DLC}^{1} = [\frac{\pi}{8}\text{rad}, 1\text{s}, 1\text{s}, 2\text{s}, 1000\text{Nm}] \qquad (14)$$

The final training horizon was set to an equidistant grid:

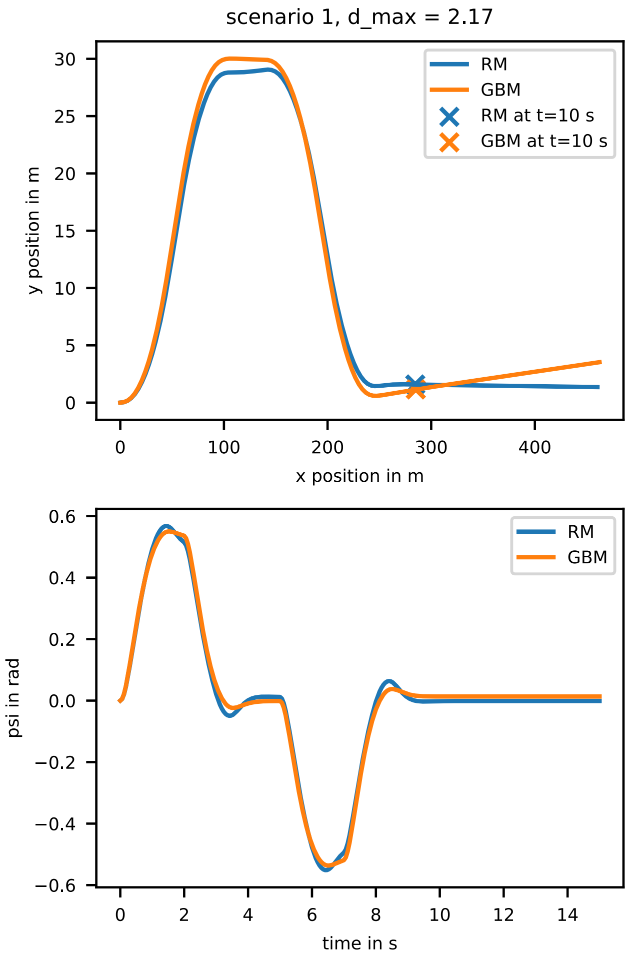


Figure 2. DLC on the training maneuver parametrized by $\boldsymbol{p}_{DLC}^{train}$

$$C_{final} = \frac{10}{32}\mathrm{s} \cdot [1,2,...,32] \tag{15}$$

C_{init}, on the other hand, held only the first three entries of C_{final}. S, the set of compared system states, was set to

$$S := \{x, y\} \tag{16}$$

so that the optimization algorithm could only optimize the ANN parameters on the basis of vehicle location data. It was **not** provided information about the vehicle yaw angle ψ. The fact that the normal load forces are very influential effects within the model and that two (scalar) functions are trained at once on only one trajectory makes this training task particularly hard.

Figure 2 displays how the GBM (after training) and the RM behave during the single training maneuver. The total simulation time was set to 15s. The top subplot of Figure 2 shows how both RM and GBM moved on the x-y plane. *Note that the axes are not scaled equally.* Both models start in the origin of the coordinate system (as specified in Equation 12). The crosses mark where both models were located at $t = 10$s, i.e. the last time at which states in S were compared to train the ANN. It can be seen how the GBM follows the track of the RM closely at first. Larger differences occur mostly after the 10s-mark. This is due to the imperfect prediction of ψ, which can be seen in the bottom plot. After the completion of the DLC, a noticeable difference remains between the predicted ψ of the GBM and that of the RM.

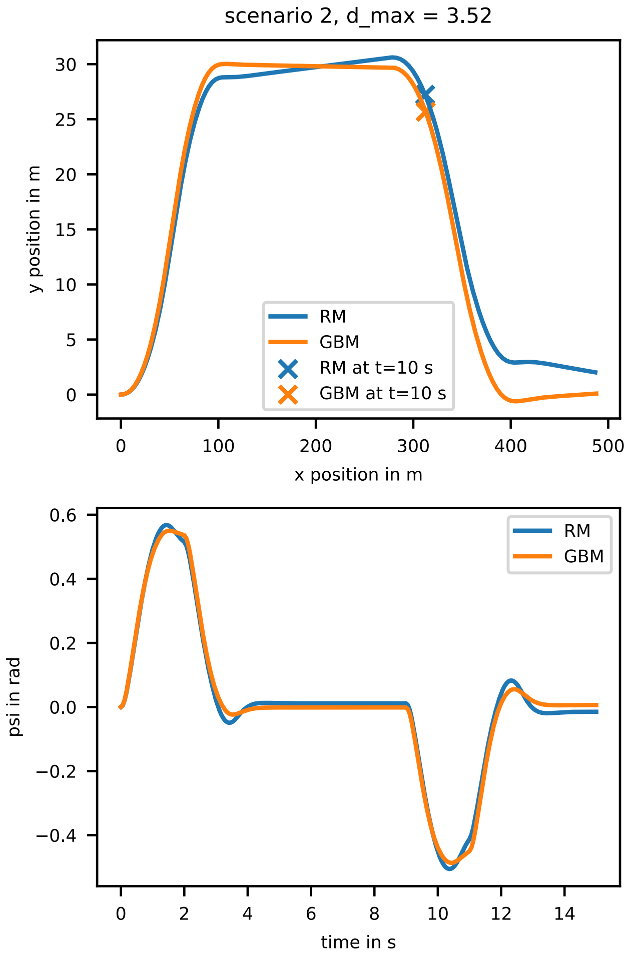


Figure 3. DLC on the training maneuver parametrized by $\boldsymbol{p}_{DLC}^{2}$

A first validation maneuver is specified in Equation 17. This scenario differs from the training scenario by the time spent on the oncoming lane. It is now three times as long.

$$\boldsymbol{p}_{DLC}^{2} = [\frac{\pi}{8}\mathrm{rad}, 1\mathrm{s}, 1\mathrm{s}, 6\mathrm{s}, 1000\mathrm{Nm}] \tag{17}$$

The results can be seen in Figure 3. This time, the largest deviations can be seen after the vehicle returns to the initial lane. Once again, the final ψ of the RM and the GBM are different so that over time, the tracks will keep drifting further apart.

Another interesting maneuver is defined in Equation 18.

$$\boldsymbol{p}_{DLC}^{3} = [\frac{\pi}{8}\mathrm{rad}, 5\mathrm{s}, 1\mathrm{s}, 2\mathrm{s}, 1000\mathrm{Nm}] \tag{18}$$

Compared to $\boldsymbol{p}_{DLC}^{1}$, it only differs by the turning time, which here is 5 times as long. This parametrization no longer looks like a DLC maneuver, but it still demonstrates how the tracks shown in Figure 4 drift apart over time.

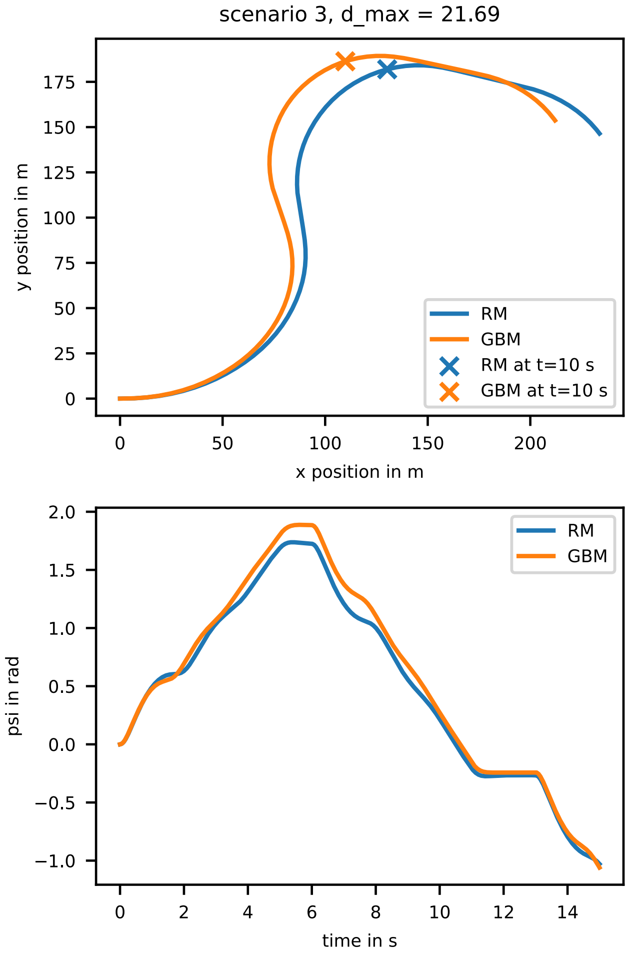


Figure 4. DLC on the training maneuver parametrized by $\boldsymbol{p}_{DLC}^{3}$

The results are heavily different for the last maneuver considered here in Equation 19.

$$\boldsymbol{p}_{DLC}^{4} = [\frac{\pi}{16}\text{rad}, 1\text{s}, 1\text{s}, 2\text{s}, 1000\text{Nm}] \qquad (19)$$

The trajectories can be seen in Figure 5. Deviations quickly build up even before changing lanes for the second time.

4.3 Discussion

Comparing (projections of) trajectories in state space in an objective manner is non-trivial. For this reason, we will resort to arguing qualitatively. Maneuver $\boldsymbol{p}_{DLC}^{1}$, $\boldsymbol{p}_{DLC}^{2}$ and $\boldsymbol{p}_{DLC}^{3}$ have in common that the tracks of both the RM and the GBM share strong similarities. Deviations become clearly visible as simulation time runs, but the behaviour seems to be comparable. Furthermore, the corresponding trajectories of ψ are encouraging since they look even more similar, although the GBM was never explicitly trained to fit its predictions along that axis of state space.

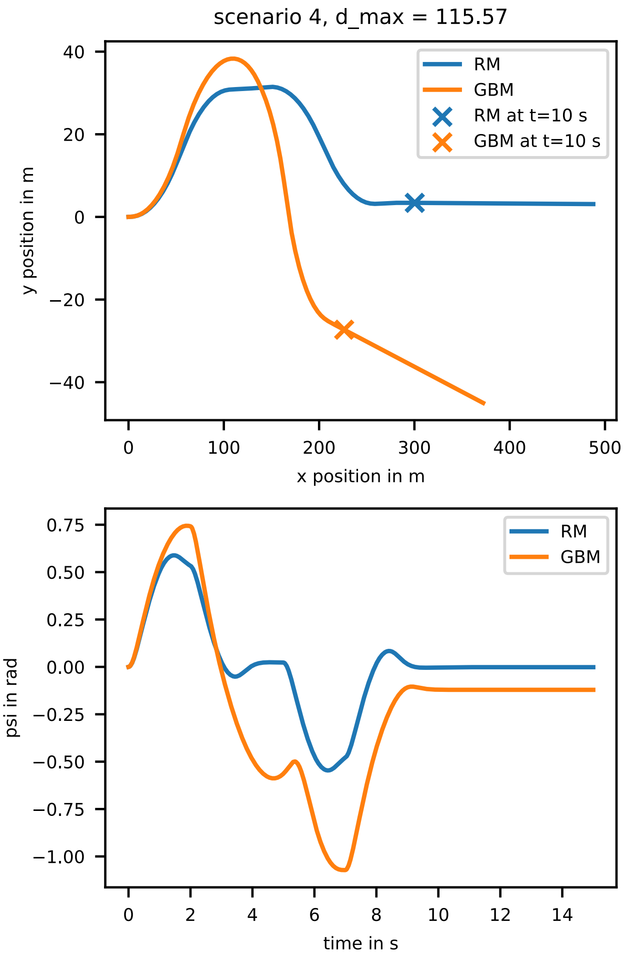


Figure 5. DLC on the training maneuver parametrized by $\boldsymbol{p}_{DLC}^{4}$

Not so in the case of $\boldsymbol{p}_{DLC}^{4}$, however. The GBM misses the track of the RM by far. This mismatch is further displayed by the corresponding trajectories of ψ.

The reason for the different results is probably the nature of the differences between the maneuver parameters. The first three maneuver parameter sets only differ in **timing** parameters (T, S, O), not signal level parameters (δ). The ANN is evaluated many times during a single maneuver simulation. Friction quantities μ_{Fx}, μ_{Fy} and μ_{Rx} are not directly influenced by the parameter sets and can vary during the simulations. This is not the case for δ. It is directly passed through to the input layer of the ANN. For this reason, during the last maneuver, the ANN was presented with inputs that it just could not have seen throughout the single training maneuver.

Although differences between GBM predictions and the data are visible, it was shown that GBM has the potential to enhance/complete white box models with very sparse data. Facilities to enhance models in this manner should become features of acausal modeling tools like Modia/TinyModia.

4.4 Remarks

l_{MSE} evaluates how well the GBM follows the track of the RM. Even if it reports a very low loss, can we ever expect our ANN to calculate 'the correct' normal forces? Or can a potentially large space of functions lead to low losses?

In order to answer this question for our trained model, we compared $\boldsymbol{f}_{X_Z}$ and $\hat{\boldsymbol{f}}_{X_Z}$ over the GBM trajectory simulated from maneuver $\boldsymbol{p}^2_{DLC}$. The results are shown in Figure 6. Note that this step cannot usually be done in practice if the underlying system equations are unknown.

As we can see, $\boldsymbol{f}_{X_Z}$ and $\hat{\boldsymbol{f}}_{X_Z}$ do not behave as similarly as we would like them to. Due to Equation 7, the components of $\boldsymbol{f}_{X_Z}$ always add up to a constant value: the reaction force that neutralises the gravitational force acting on the vehicle CoM. When we replaced the set of equations by the ANN, we explicitly neglected this invariance. And as we can see, our optimization technique did not recover said invariance from the single training trajectory.

5 Framework of Tools

5.1 GBM

The model that was described in 2.1 has been declared with the help of TinyModia. The model inputs are passed to the vehicle model using a separate 'Driver' model that sets the inputs.

The actual replacement of Equations 8 to 7 was done by modifying the AST of the model directly before the model instantiation process mentioned in subsection 1.4: $\boldsymbol{f}_{X_Z}$ was replaced by $\hat{\boldsymbol{f}}_{X_Z}$. During said process, the `getDerivatives!` method was written to a file and later modified *by hand* as detailed in the next subsection. With the second feature in 5.3, this process would have been easy to automate as well.

5.2 Training Procedure

As a part of a gradient descent based optimization of $l_{MSE}(\boldsymbol{\theta})$, gradients/sensitivities of $\hat{\boldsymbol{u}}(t)$ w.r.t. $\boldsymbol{\theta}$ are required. For this task, we relied on `Zygote.jl` (Mike Innes et al. 2019) and `DifferentialEquations.jl` (Rackauckas and Nie 2017).

`DifferentialEquations`'s `solve` was used with solver `Tsit5()` and default options to simulate the GBM in order to yield $\hat{\boldsymbol{u}}(t)$. To differentiate $\hat{\boldsymbol{u}}(t)$ at specific points w.r.t. $\boldsymbol{\theta}$, `Zygote` needs information on how to calculate sensitivities of ODE solutions. This gap is filled using package `DiffEqSensitivity.jl`. In the standard setting, which we used, an approach based on adjoint sensitivities is used.

`DifferentialEquations`'s `solve` requires the user to specify an array of parameters with respect to which the ODE solution can then be differentiated.

At the time of writing this, this detail may be inconvenient to users. It is the reason why the results returned from TinyModia's `simulate!` calls can not be differentiated w.r.t. model parameters.

We worked around this issue by further modifying the `getDerivatives!` function mentioned in subsection 5.1. We made $\boldsymbol{\theta}$ a function parameter as well and then called `DifferentialEquations`'s `solve` ourselves to enable differentiation.

5.3 Requested Features in Modia/TinyModia

To streamline the GBM workflow, we would like to see the following features in Modia/TinyModia:

- **Compatibility of `simulate!` with `Zygote.jl`**
  The need for a lot of work currently neccessary in order to generate GBM would be removed if Modia/TinyModia simulation results could be differentiated with respect to model parameters. Since `simulate!` internally uses `DifferentialEquations`'s `solve` that generally offers this functionality, we believe that the main work to be done here resides in the creation of a serialized version of all model parameters (i.e. in a single Array passed to `solve`).

- **The ability to obtain the AST of `getDerivatives!` conveniently after instantiation**
  It should not be neccessary to print the code of `getDerivatives!` to file from a debug log. Instead, the AST should be returned as an expression if the user requests this. This feature would be very handy for the more experimental case in which the user does not know beforehand where to best place ANN.

- **The ability to mark equations for later replacement by ANN-enhanced equations**
  We modified assignments manually with ANN in the generated simulation code.
  Normally, users of software intended for acausal modeling and subsequent simulation do not interact with intermediate representations of the model equations or the generated simulation code. They may have a slight idea of which of the model components in their model is "faulty" and leads to errors observed when simulation results are compared to real-world measurements. Take air drag as an example. Drag is a complex phenomenon and it is unlikely that a model as simple as ours in Equation 4 is sufficient to make good predictions in a real-world scenario with very low and high vehicle velocities. We are certain about which are the "faulty" model equations but we would not know how to improve them without extensive knowledge about aerodynamics.

  However in order to insert ANN into those "faulty" equations like we did in this paper, a user would have to modify the simulation code by hand like we did. They may have trouble finding the exact lines of code produced from their "faulty" component equations.

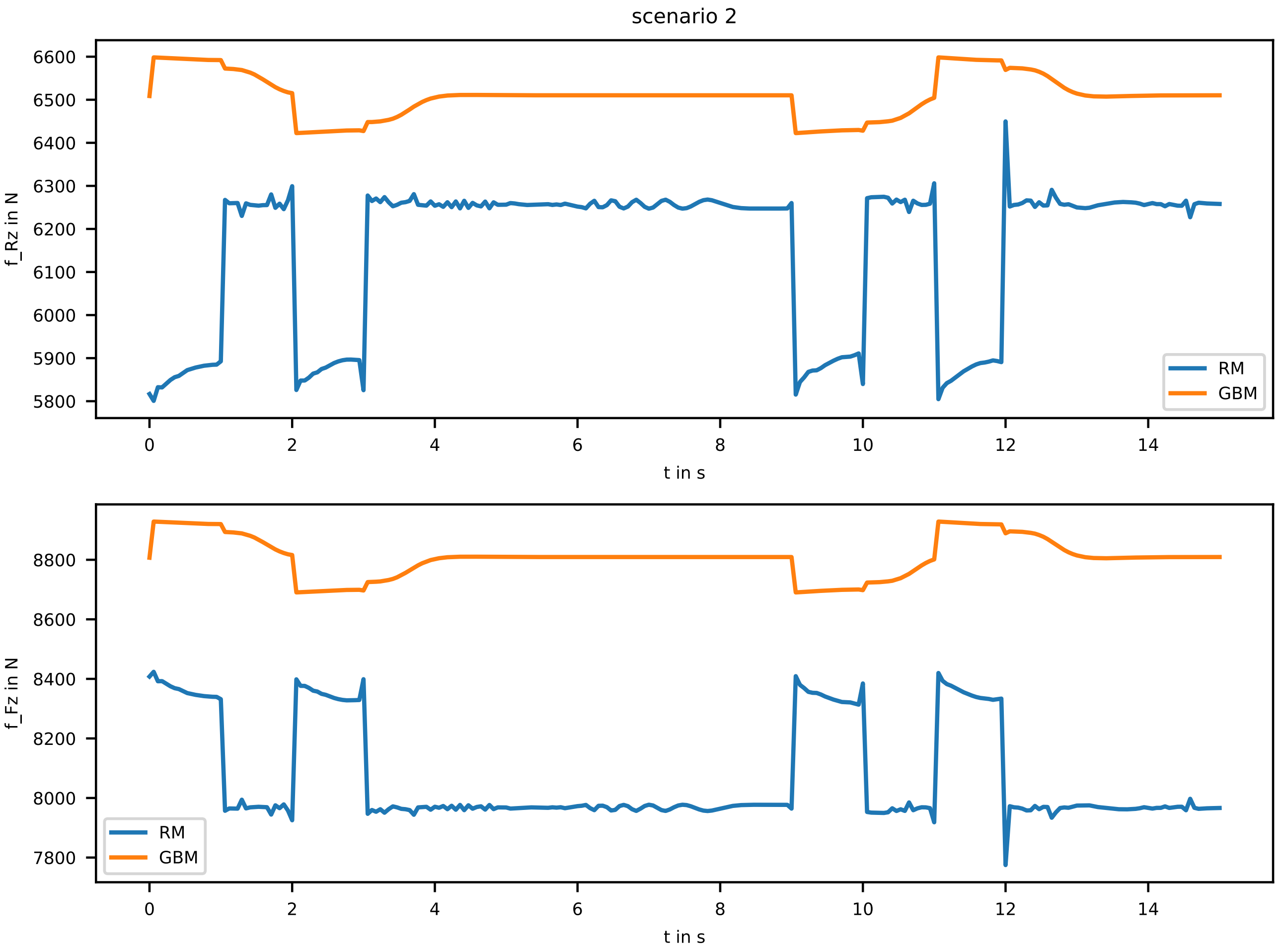


Figure 6. Comparison of the normal forces. The RM internally uses $\boldsymbol{f}_{X_z}$ to calculate normal forces f_{Rz} and f_{Fz}, the GBM uses $\hat{\boldsymbol{f}}_{X_z}$.

Even then they are restricted to the causalization created by the model transformation algorithm. After all, the user might want to include new variables in those equations.

We would like to have a feature that lets the user mark equations they are uncertain about with a tag and possibly additional desired involved model variables during the modeling process. The causalization algorithm would then work as usual and determine a sequence of calculation and possibly algebraic loops whilst keeping track of the user-provided equation tags. The user would then be informed about where and how their equations are used in the final calculation graph. Whether they are part of an algebraic loop, what their (or the corresponding loop's) inputs are and which model variable they were matched to.

With this information, the user can create an ANN with the right input/output size. The user would be very free to design what happens in the ANN: It could just be a densely connected layers with appropriate activation functions. Or it could take the original "faulty" equation as a basis and just add such a network so that the latter only has to learn a difference.

After the user has provided an ANN for each of the marked equations, a code generating algorithm would produce the actual simulation code for the model. This then leaves the user with an ANN-infused model that can be trained on an arbitrary data set.

6 Future Research

We are planning to experiment with different training schemes that operate on a set of trajectories instead of just a single training maneuvers.

Moreover, it may be beneficial to alter the way we build time horizons over which we compare trajectories. Instead of growing a single horizon to full length, one could slice a single trajectory into several segments and calculate gradients in parallel.

Another issue that needs to be addressed is the normalisation/transformation of the inputs that reach the ANN parts of the model. Some neural network architectures/activation functions were designed assuming that inputs to the input layer follow specific distributions. We did not account for this yet because it is an inherent property of our model structure. Before we simulate our model, we do not yet know what the inputs to the ANN will be. After all, they depend on the solution to the ODE. It should be possible to construct an algorithm that adapts a prescaler to keep inputs over training trajectories inside specific bounds. Future research will have to show whether this is feasible or whether this situation can be solved differently e.g. by making use of different activation functions.

For our loss function, we relied on a very simple MSE-based formulation. Other (differentiable) metrics such as soft dynamic time warping (Cuturi and Blondel 2017) exist in order to compare time series data. We will evaluate whether these alternative metrics are beneficial to our training results.

Furthermore, we will examine whether invertible neural networks as in (Ardizzone et al. 2019) can help overcome the issue of ANN ending up in inefficient root-finding loops.

7 Conclusion

We demonstrated that GBM can be trained with very sparse data to yield remarkable results. Furthermore, we detailed how to achieve this with free software and made suggestions for features that would make this process a lot simpler for the user.

References

Ardizzone, Lynton et al. (2019). *Analyzing Inverse Problems with Invertible Neural Networks*. arXiv: 1808.04730 [cs.LG].

Bezanson, Jeff et al. (2012). “Julia: A fast dynamic language for technical computing”. In: *arXiv preprint arXiv:1209.5145*.

Cuturi, Marco and Mathieu Blondel (2017). “Soft-dtw: a differentiable loss function for time-series”. In: *International Conference on Machine Learning*. PMLR, pp. 894–903.

Elmqvist, Hilding and Martin Otter (2017). “Innovations for future Modelica”. In: *Proceedings of 12th International Modelica Conference*. Linköping University Electronic Press.

Elmqvist, Hilding and Martin Otter (2021). *TinyModia*. https://github.com/ModiaSim/TinyModia.jl. Accessed: 2021-05-07.

Innes, Michael et al. (2018). “Fashionable Modelling with Flux”. In: *CoRR* abs/1811.01457. arXiv: 1811.01457. URL: https://arxiv.org/abs/1811.01457.

Innes, Mike et al. (2019). “A differentiable programming system to bridge machine learning and scientific computing”. In: *arXiv preprint arXiv:1907.07587*.

Karpatne, Anuj et al. (2018). *Physics-guided Neural Networks (PGNN): An Application in Lake Temperature Modeling*. arXiv: 1710.11431 [cs.LG].

Ma, Yingbo et al. (2021). *ModelingToolkit: A Composable Graph Transformation System For Equation-Based Modeling*. arXiv: 2103.05244 [cs.MS].

Pacejka, Hans (2005). *Tire and vehicle dynamics*. Elsevier.

Rackauckas, Christopher, Yingbo Ma, et al. (2020). *Universal Differential Equations for Scientific Machine Learning*. arXiv: 2001.04385 [cs.LG].

Rackauckas, Christopher and Qing Nie (2017). “Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia”. In: *Journal of Open Research Software* 5.1.

Rai, R. and C. K. Sahu (2020). “Driven by Data or Derived Through Physics? A Review of Hybrid Physics Guided Machine Learning Techniques With Cyber-Physical System (CPS) Focus”. In: *IEEE Access* 8, pp. 71050–71073.

Velenis, Efstathios, Emilio Frazzoli, and Panagiotis Tsiotras (2009). “On steady-state cornering equilibria for wheeled vehicles with drift”. In: *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. IEEE, pp. 3545–3550.

Willard, Jared et al. (2020). *Integrating Physics-Based Modeling with Machine Learning: A Survey*. arXiv: 2003.04919 [physics.comp-ph].

Composing Modeling and Simulation with Machine Learning in Julia

Chris Rackauckas[1,2] Ranjan Anantharaman[2] Alan Edelman[2] Shashi Gowda[2] Maja Gwozdz[1] Anand Jain[1] Chris Laughman[3] Yingbo Ma[1] Francesco Martinuzzi[1] Avik Pal[1] Utkarsh Rajput[1] Elliot Saba[1] Viral B. Shah[1]

[1]Julia Computing Inc., USA
[2]Massachusetts Institute of Technology, USA
[3]Mitsubishi Electric Research Lab, USA

Abstract

In this paper we introduce JuliaSim, a high-performance programming environment designed to blend traditional modeling and simulation with machine learning. JuliaSim can build accelerated surrogates from component-based models, such as those conforming to the FMI standard, using continuous-time echo state networks (CTESN). The foundation of this environment, ModelingToolkit.jl, is an acausal modeling language which can compose the trained surrogates as components within its staged compilation process. As a complementary factor we present the JuliaSim model library, a standard library with differential-algebraic equations and pre-trained surrogates, which can be composed using the modeling system for design, optimization, and control. We demonstrate the effectiveness of the surrogate-accelerated modeling and simulation approach on HVAC dynamics by showing that the CTESN surrogates accurately capture the dynamics of a HVAC cycle at less than 4% error while accelerating its simulation by 340x. We illustrate the use of surrogate acceleration in the design process via global optimization of simulation parameters using the embedded surrogate, yielding a speedup of two orders of magnitude to find the optimum. We showcase the surrogate deployed in a co-simulation loop, as a drop-in replacement for one of the coupled FMUs, allowing engineers to effectively explore the design space of a coupled system. Together this demonstrates a workflow for automating the integration of machine learning techniques into traditional modeling and simulation processes.

Keywords: modeling, simulation, Julia, machine learning, surrogate modeling, acceleration, co-simulation, Functional Mock-up Interface

1 Introduction

With the dramatic success of artificial intelligence and machine learning (AI/ML) throughout many disciplines, one major question is how AI/ML will change the field of modeling and simulation. Modern modeling and simulation involves the time integration of detailed multiphysics component models, programmatically generated by domain-specific simulation software. Their large computational expense makes design, optimization and control of these systems prohibitively expensive (Benner, Gugercin, and Willcox 2015). Thus one of the major proposed avenues for AI/ML in the space of modeling and simulation is in the generation of reduced models and data-driven surrogates, that is, sufficiently accurate approximations with majorly reduced computational burden (Willard et al. 2020; Ratnaswamy et al. 2019; Zhang et al. 2020; Y. Kim et al. 2020; Hu et al. 2020). While the research has shown many cross-domain successes, the average modeler does not employ surrogates in most projects for a number of reasons: the surrogatization process is not robust enough to be used blindly, it can be difficult to ascertain whether the surrogate approximation is sufficiently accurate to trust the results, and it is not automated in modeling languages. This begs the question – how does one develop a modeling environment that seamlessly integrates traditional and machine learning approaches in order to merge this newfound speed with the robustness of stabilized integration techniques?

The difficulty of addressing these questions comes down to the intricate domain-specific algorithms which have been developed over the previous decades. Many scientists and engineers practice modeling and simulation using acausal modeling languages, which require sophisticated symbolic algorithms in order to give a stable result. Algorithms, such as alias elimination (Otter and Elmqvist 2017) and the Pantelides algorithm for index reduction (Pantelides 1988), drive the backend of current Modelica compilers like Dymola (Brück et al. 2002) and OpenModelica (Fritzson, Aronsson, et al. 2005) and allow for large-scale differential-algebraic equation (DAE) models to be effectively solved. Notably, these compiler pipelines encode exact symbolic transformations. One can think of generalizing this process by allowing approximate symbolic transformations, which can thus include model reduction and machine learning techniques. As this process now allows for inexact transformation, the modeling language would need to allow users to interact with the compiler. Moreover, it would have to allow users to swap in and out approximations, selectively accelerate specific

DOI
10.3384/ecp2118197

submodels, and finally make it easy to check the results against the non-approximated model.

To address these issues, we introduce JuliaSim — a modeling and simulation environment, which merges elements of acausal modeling frameworks like Modelica with machine learning elements. The core of the environment is the open source ModelingToolkit.jl (Ma et al. 2021), an acausal modeling framework with an interactive compilation mechanism for including exact and inexact transformations. To incorporate machine learning, we describe the continuous-time echo state network (CTESN) architecture as an approximation transformation of time series data to a DAE component. Notably, the CTESN architecture allows for an implicit training to handle the stiff equations common in engineering simulations. To demonstrate the utility of this architecture, we showcase the CTESN as a methodology for translating a Room Air Conditioner model from a Functional Mock-up Unit (FMU) binary to an accelerated ModelingToolkit.jl [1] model with 4% error over the operating parameter range, accelerating it by 340x. We then show how the accelerated model can be used to speed up global parameter optimization by over two orders of magnitude. As a component within an acausal modeling framework, we demonstrate its ability to be composed with other models, here specifically in the context of the FMI co-simulation environment. We believe that these results indicate the promise of blending machine learning surrogate methods in the broader modeling and simulation workflow.

2 Overview of JuliaSim

The flow of the architecture (Figure 1) is described as follows. We start by describing the open ModelingToolkit.jl acausal modeling language as a language with composable transformation passes to include exact and approximate symbolic transformations. To incorporate machine learning into this acausal modeling environment, we describe the CTESN, which is a learnable DAE structure that can be trained on highly stiff time series to build a representation of a component. To expand the utility of components, we outline the interaction with the FMI standard to allow for connecting and composing models. Finally, we present the JuliaSim model library, which is a collection of acausal components that includes pre-trained surrogates of models so that users can utilize the acceleration without having to pay for the cost of training locally.

2.1 Interactive Acausal Modeling with ModelingToolkit.jl

ModelingToolkit.jl (Ma et al. 2021) (MTK) is a framework for equation-based acausal modeling written in the Julia programming language (Bezanson et al. 2017), which generates large systems of DAEs from symbolic models. MTK takes a different approach than Modia.jl[2], another Julia package for acausal modeling. For a comparison between MTK, Modia and Modelica, the reader referred to this article [3] as well this section of the documentation [4]. Similarly to Modelica, MTK allows for building models hierarchically in a component-based fashion. For example, defining a component in MTK is to define a function which generates an ODESystem:

```
function Capacitor(;name, C = 1.0)
    val = C
    @named p = Pin(); @named n = Pin()
    @variables v(t); @parameters C
    D = Differential(t)
    eqs = [v ~ p.v - n.v
           0 ~ p.i + n.i
           D(v) ~ p.i / C]
    ODESystem(eqs, t, [v], [C],
        systems=[p, n],
        defaults=Dict(C => val),
        name=name)
end
```

Systems can then be composed by declaring subsystems and defining the connections between them. For instance, the classic RC circuit can be built from standard electrical components as:

```
@named resistor = Resistor(R=100)
@named capacitor = Capacitor(C=0.001)
@named source = ConstantVoltage(V=10)
@named ground = Ground()
@named rc_model = ODESystem([
    connect(source.p, resistor.p)
    connect(resistor.n, capacitor.p)
    connect(capacitor.n, source.n,
            ground.g)],
    t, systems=[resistor, capacitor,
                source, ground])
```

The core of MTK's utility is its system of transformations, where a transformation is a function which takes an `AbstractSystem` type to another `AbstractSystem` type. Given this definition, transformations can be composed and chained. Transformations, such as `dae_index_lowering`, transform a higher-index DAE into an index-1 DAE via the Pantelides algortithm (Pantelides 1988). Nonlinear tearing and `alias_elimination` (Otter and Elmqvist 2017) are other commonly used transformations, which match the workflow of the Dymola Modelica compiler (Brück et al. 2002) (and together are given the alias `structural_simplify`). However, within this system the user can freely compose transformations with domain- and problem-specific transformations, such as "exponentiation of a variable to enforce positivity" or "extending the system to include the tangent space". After

[1] https://github.com/SciML/ModelingToolkit.jl

[2] https://github.com/ModiaSim/Modia.jl

[3] http://www.stochasticlifestyle.com/modelingtoolkit-modelica-and-modia-the-composable-modeling-future-in-julia/

[4] https://mtk.sciml.ai/stable/comparison/

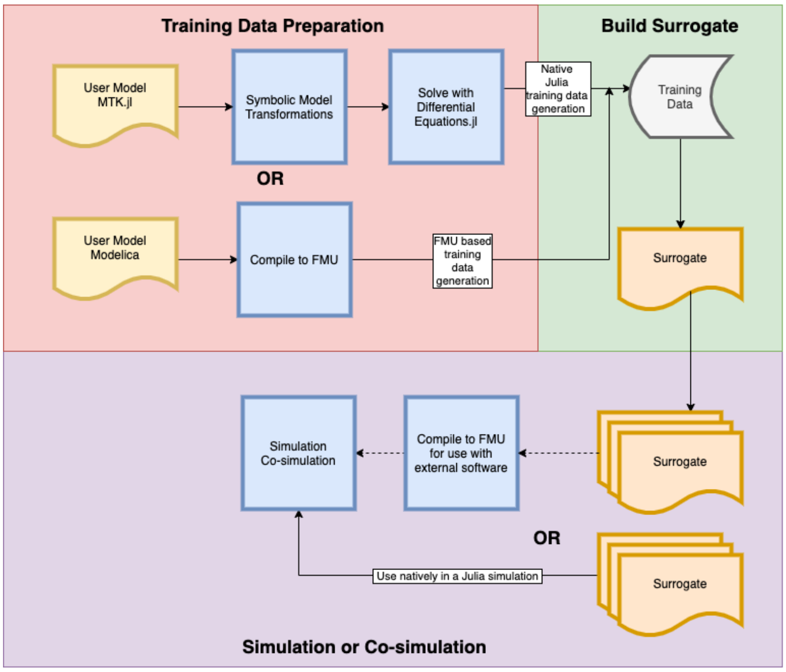

Training Data Preparation
Build Surrogate
User Model MTK.jl
Symbolic Model Transformations
Solve with Differential Equations.jl
Native Julia training data generation
Training Data
OR
User Model Modelica
Compile to FMU
FMU based training data generation
Surrogate
Simulation Co-simulation
Compile to FMU for use with external software
Surrogate
OR
Use natively in a Julia simulation
Surrogate
Simulation or Co-simulation

Figure 1. Compiler passes in the JuliaSim Modeling and Simulation system. Ordinarily, most systems simulate equation-based models, described in the "Training Data Preparation" and the "Simulation or Co-simulation" phases. We provide an additional set of steps in our compiler to compute surrogates of models. Blue boxes represent code transformations, yellow represents user source code, gray represents data sources, and gold represents surrogate models. The dotted line indicates a feature that is currently work in progress.

transformations have been composed, the `ODEProblem` constructor compiles the resulting model to a native Julia function for usage with DifferentialEquations.jl (Rackauckas and Nie 2017).

2.2 Representing Surrogates as DAEs with Continuous-Time Echo State Networks

In order to compose a trained machine learning model with the components of ModelingToolkit.jl, one needs to represent such a trained model as a set of DAEs. To this end, one can make use of continuous machine learning architectures, such as neural ODEs (Chen et al. 2018) or physics-informed neural networks (Raissi, Perdikaris, and Karniadakis 2019). However, prior work has demonstrated that such architectures are prone to instabilities when being trained on stiff models (Wang, Teng, and Perdikaris 2020). In order to account for these difficulties, we have recently demonstrated a new architecture, CTESNs, which allows for implicit training in parameter space to stabilize the ill-conditioning present in stiff systems (Anantharaman et al. 2021). For this reason, CTESNs are the default surrogate algorithm of JuliaSim and will be the surrogate algorithm used throughout the rest of the paper. We provide an overview of the CTESN here, but for more details on the method, we refer the reader to (Anantharaman et al. 2021).

The CTESN is a continuous-time generalization of echo state networks (ESNs) (Lukoševičius 2012), a reservoir computing framework for learning a nonlinear map by projecting the inputs onto high-dimensional spaces through predefined dynamics of a nonlinear system (Lukoševičius and Jaeger 2009). CTESNs are effective at learning the dynamics of systems with widely separated time scales because their design eliminates the requirement of training via local optimization algorithms, like gradient descent, which are differential equation solvers in a stiff parameter space. Instead of using optimization, CTESNs are semi-implicit neural ODEs where the first layer is fixed, which results in an implicit training process.

To develop the CTESN, first a non-stiff dynamical system, called the reservoir, is chosen. This is given by the expression

$$r' = f\left(Ar + W_{hyb}x\left(p^*, t\right)\right) \tag{1}$$

where A is a fixed random sparse $N_R \times N_R$ matrix, W_{hyb} is a fixed random dense $N_R \times N$ matrix, and $x(p^*, t)$ is a solution of the system at a candidate set of parameters from the parameter space, and f is an activation function.

Projections (W_{out}) from the simulated reservoir time series to the truth solution time series are then computed, using the following equation:

$$x(t) = g\left(W_{out}\, r(t)\right) \tag{2}$$

where g is an activation function (usually the identity), $r(t)$ represents the solution to the reservoir equation, and $x(t)$ represents the solution to full model. This projection is usually computed via least-squares minimization using the singular value decomposition (SVD), which is robust to ill-conditioning by avoiding gradient-based optimization. A projection is computed for each point in the parameter space, and a map is constructed from the parameter space P to each projection matrix W_{out} (in our examples, we will use a radial basis function to construct this map). Thus our final prediction is the following:

$$\hat{x}(t) = g(W_{out}(\hat{p})r(t)) \tag{3}$$

For a given test parameter $\hat{p}$, a matrix $W_{out}(\hat{p})$ is computed, the reservoir equation is simulated, and then the final prediction $\hat{x}$ is a given by the above matrix multiplication.

While the formulation above details linear projections from the reservoir time series (Linear Projection CTESN or LPCTESN), nonlinear projections in the form of parametrized functions can also be used to project from the reservoir time series to the reference solution (Nonlinear Projection CTESN). For this variation, a radial basis function can be applied to model the nonlinear projection $r(t) \mapsto x(t)$ in equation 2. The learned polynomial coefficients β_i from radial basis functions are used, and a mapping between the model parameter space and coefficients β_i's is constructed.

$$\text{rbf}(\beta_i)(r(t)) \approx x(p_i,t) \quad \forall i \in \{1,\ldots,k\} \tag{4}$$

$$\text{rbf}(p_i) \approx \beta_i \quad \forall i \in \{1,\ldots,k\} \tag{5}$$

where k is the total number of parameter samples used for training. Finally, during prediction, first the coefficients are predicted and a radial basis function for the prediction of the time series is constructed:

$$\hat{\beta} = \text{rbf}(\hat{p}) \tag{6}$$

$$\hat{x}(t) = \text{rbf}(\hat{\beta})(r(t)) \tag{7}$$

Notice that both the LPCTESN and the NPCTESN represent the trained model as a set of DAEs, and thus can be represented as an `ODESystem` in MTK, and can be composed similarly to any other DAE model.

A significant advantage of applying NPCTESNs over LPCTESNs is the reduction of reservoir sizes, which creates a cheaper surrogate with respect to memory usage. LPCTESNs often use reservoirs whose dimensions reach an order of 1000. While this reservoir ODE is not-stiff, and is cheap to simulate, this leads to higher memory requirements. Consider the surrogatization of the Robertson equations (Robertson 1976), a canonical stiff benchmark problem:

$$\dot{y_1} = -0.04y_1 + 10^4 y_2 \cdot y_3 \tag{8}$$

$$\dot{y_2} = 0.04y_1 - 10^4 y_2 \cdot y_3 - 3 \cdot 10^7 y_2^2 \tag{9}$$

$$\dot{y_3} = 3 \cdot 10^7 y_2^2 \tag{10}$$

where y_1, y_2, and y_3 are the concentrations of three reagants. This system has widely separated reaction rates $(0.04, 10^4, 3 \cdot 10^7)$, and is well-known to be very stiff (Gobbert 1996; Robertson and Williams 1975; Robertson 1976). It is commonly used as an example for evaluating integrators of stiff ODEs (Hosea and Shampine 1996). Finding an accurate surrogate for this system is difficult because it needs to capture both the stable slow-reacting system and the fast transients. This breaks many data-driven surrogate methods, such as PINNs and LSTMs (Anantharaman et al. 2021). We shall now demonstrate training a surrogate of this system with the reaction rates as inputs/design parameters.

Table 1 shows the result of surrogatization using the LPCTESN and the NPCTESN, while considering the following ranges of design parameters corresponding to the three reaction rates: $(0.036, 0.044)$, $(2.7 \cdot 10^7, 3.3 \cdot 10^7)$ and $(0.9 \cdot 10^4, 1.1 \cdot 10^4)$. We observe three orders of magnitude smaller reservoir equation size, resulting in a computationally cheaper surrogate model.

Table 1. Comparison between LPCTESN and NPCTESN on surrogatization of the Robertson equations. "Res" stands for reservoir.

Model	*Res. ODE size*	*Avg Rel. Err %*
LPCTESN	3000	0.1484
NPCTESN	3	0.0200

2.3 Composing with External Models via the FMI Standard

While these surrogatized CTESNs can be composed with other MTK models, more opportunities can be gained by composing with models from external languages. The Functional Mock-up Interface (FMI) (Blochwitz et al. 2011) is an open-source standard for coupled simulation, adopted and supported by many simulation tools[5], both open source and commercial. Models can be exported as *Functional Mock-up Units* (FMUs), which can then be simulated in a shared environment. Two forms of coupled simulation are standardized. *Model exchange* uses a centralized time-integration algorithm to solve the coupled sets of differential-algebraic equations exported by the individual FMUs. The second approach, *co-simulation*, allows FMUs to export their own simulation routine, and synchronizes them using a master algorithm. Notice that as DAEs, the FMU interface is compatible with ModelingToolkit.jl components and, importantly, trained CTESN models.

JuliaSim can simulate an FMU in parallel at different points in the design space. For each independent simulation, the `fmpy` package[6] was used to run the FMU in ModelExchange with CVODE (Cohen, Hindmarsh, and Dubois 1996) or co-simulation with the FMUs exported

[5] https://fmi-standard.org/tools/

[6] https://github.com/CATIA-Systems/FMPy

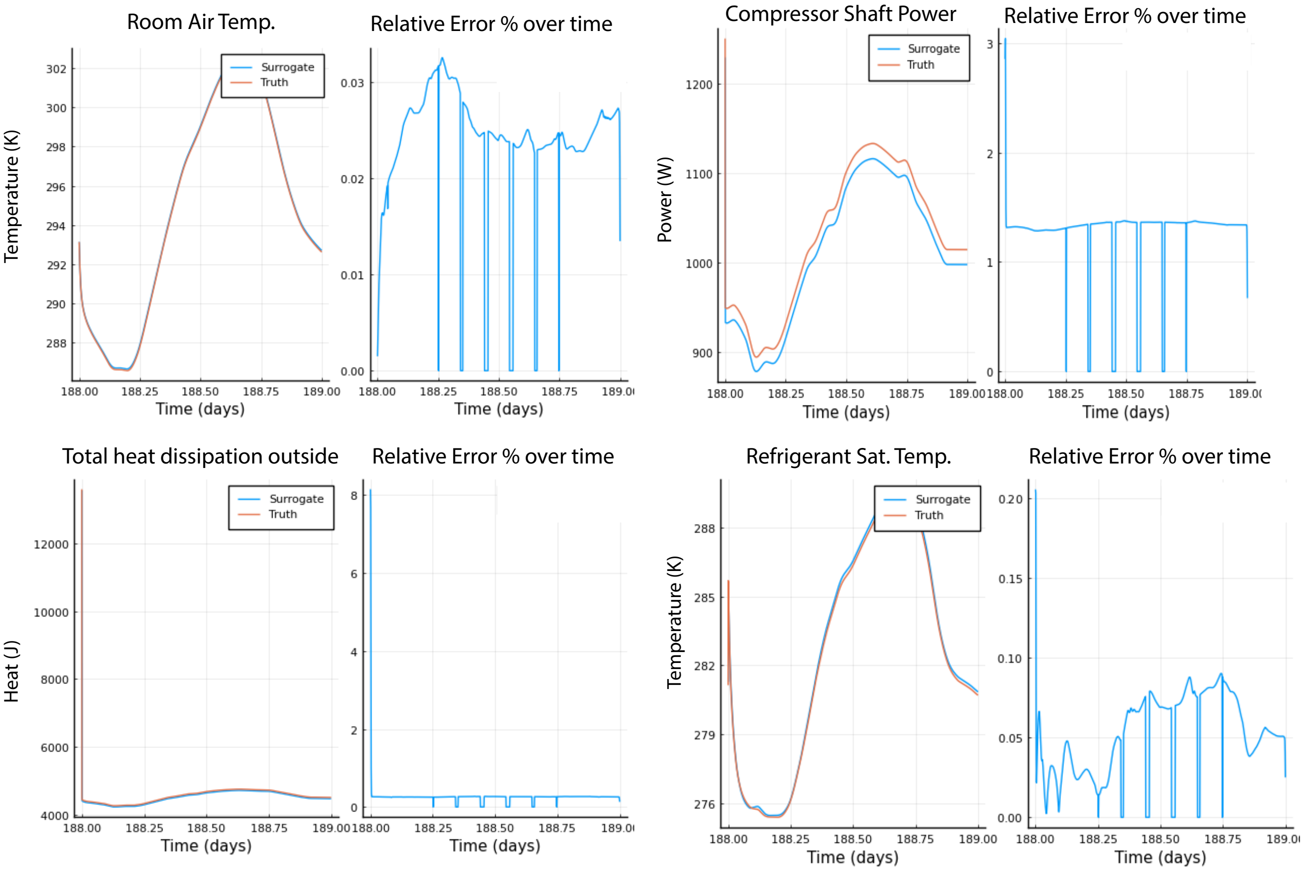


Figure 2. Surrogate prediction of the room temperature of the RAC model in blue, while the ground truth is in red. This is a prediction for points over which the surrogate has not been trained. Relative error is calculated throughout the time span at 1000 uniformly spaced points. The CTESN surrogate was trained on a timespan of an entire day, using data from 100 simulations. The simulation parameters were sampled from a chosen input space using Latin hypercube sampling. The simulation time span goes from 188 days to 189 days at a fixed step size of 5 seconds. Table 3 presents the list of and ranges of inputs the surrogate has been trained on. The relative error usually peaks at a point with a discontinuous derivative in time, usually induced by a step or ramp input (which, in this case, is the parametrized compressor speed ramp input.). Another feature of the prediction error above is that it is sometimes stable throughout the time span (such as with the compressor shaft power, top right). This is a feature of how certain outputs vary through the parameter space. Sampling the space with more points or reducing the range of the chosen input space would reduce this error. Table 2 shows the maximum relative error computed for many other outputs of interest. Figure 3 computes and aggregates maximum errors across a 100 new test points from the space.

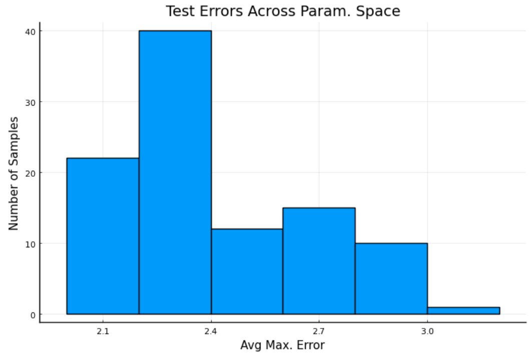


Figure 3. Performance of surrogate when tested on 100 test parameters from the parameter space. The test parameters were chosen via Sobol low discrepancy sampling, and maximum relative error across the time span was calculated for all output quantities. The average maximum error across all output quantities was then plotted as a histogram. Our current test points may not be maximally separated through the space, but we anticipate similar performance with more test examples and a maximal sampling scheme.

Table 2. Relative errors when the surrogate is tested on parameters it has not been trained on. HEX stands for "heat exchanger" and LEV stands for "linear expansion valve".

Output quantity	*Max. Rel. Err %*	*Output quantity*	*Max. Rel. Err %*
Air temp. in room	0.033	Rel. humidity in room	0.872
Outdoor dry bulb temp.	0.0001	Outdoor rel. humidity	0.003
Compressor inlet pressure	4.79	Compressor outlet pressure	3.50
LEV inlet pressure	3.48	LEV outlet pressure	4.84
LEV refrigerant outlet enthalpy	1.31	Compressor refrigerant mass flow rate	4.51
Evaporator refrigerant saturation temp.	0.205	Evaporator refrigerant outlet temp.	0.145
Total heat dissipation of outdoor HEX	8.15	Sensible heat load of indoor HEX	0.892
Latent heat load of indoor HEX	3.51	Outdoor coil outlet air temperature	0.432
Indoor coil outlet air temperature	0.070	Compressor shaft power	3.04

Table 3. Surrogate Operating Parameters. The surrogate is expected to work over this entire range of design parameters.

Input	*Parameter Range*
Compressor Speed (ramp)	Start Time - (900, 1100) s Start Value - (45, 55) rpm Offset - (9, 11) rpm
LEV Position	(252, 300)
Outdoor Unit Fan Speed	(680, 820) rpm
Indoor Unit Fan Speed	(270, 330) rpm
Radiative Heat Gain	(0.0, 0.1)
Convective Heat Gain	(0.0, 0.1)
Latent Heat Gain	(0.3, 0.4)

solver. The resultant time series was then fitted to cubic splines. Integration with state-of-the-art solvers from DifferentialEquations.jl (Rackauckas and Nie 2017) for simulating ModelExchange FMUs is planned in future releases.

2.4 Incorporating Surrogates into the JuliaSim Model Library

Reduced order modeling and surrogates in the space of simulation have traditionally targeted PDE problems because of the common reuse of standard PDE models such as Navier-Stokes equations. Since surrogates have a training cost, it is only beneficial to use them if that cost is amortized over many use cases. In equation-based modeling systems, such as Modelica or Simulink, it is common for each modeler to build and simulate a unique model. While at face value this may seem to defeat opportunities for amortizing the cost, the composability of components within these systems is what grants a new opportunity. For example, in Modelica it is common to hierarchically build models from components originating in libraries, such as the Modelica standard library. This means that large components, such as high-fidelity models of air conditioners, specific electrical components, or physiological organelles, could be surrogatized and accelerate enough workflows to overcome the training cost[7]. In addition, if the modeler is presented with both the component and its pre-trained surrogate with known accuracy statistics, such a modeler could effectively use the surrogate (e.g., to perform a parameter study) and easily swap back to the high- fidelity version for the final model. This allows users to test the surrogate in their downstream application, examine the resulting behaviour, and make a decision on whether the surrogate is good enough for their task. A discussion of error dynamics of the surrogate is left to future work.

Thus to complement the JuliaSim surrogatization architecture with a set of pre-trained components, we developed the JuliaSim Model Library and training infrastructure for large-scale surrogatization of DAE models. JuliaSim's automated model training pipeline can serve and store surrogates in the cloud. It consists of models from the Modelica Standard Library, CellML Physiome model repository (Yu et al. 2011), and other benchmark problems defined using ModelingToolkit. In future work, we shall demonstrate workflows using these surrogates for accelerated design and development.

Each of the models in the library contains a source form which is checked by continuous integration scripts, and surrogates are regenerated using cloud resources whenever the source model is updated[8]. For some models, custom importers are also run in advance of the surrogate generation. For instance, the CellMLToolkit.jl importer translates the XML-based CellML schema into ModelingToolkit.jl. Components and surrogates from other sources, such as Systems Biology Markup Language libraries (SBML), are scheduled to be generated. Additionally, for each model, a diagnostic report is generated detailing:

1. the accuracy of the surrogate across all outputs of interest

[7] We note that an additional argument can be made for pre-trained models in terms of user experience. If a user of a modeling software needs a faster model for real-time control, then having raised the total simulation cost to reduce the real-time user cost would still have a net benefit in terms of the application

[8] https://buildkite.com/

2. the parameter space which was trained on

3. and performance of the surrogate against the original model

is created to be served along with the models. With this information, a modeler can check whether the surrogatized form matches the operating requirements of their simulation and replace the usage of the original component with the surrogate as necessary. Note that a GUI exists for users of JuliaSim to surrogatize their own components through this same system.

3 Accelerating Building Simulation with Composable Surrogates

To demonstrate the utility of the JuliaSim architecture, we focus on accelerating the simulation of energy efficiency of buildings. Sustainable building simulation and design involves evaluating multiple options, such as building envelope construction, Heating Ventilation, Air Conditioning and Refrigeration (HVAC/R) systems, power systems and control strategies. Each choice is modeled independently by specialists drawing upon many years of development, using different tools, each with their own strengths (Wetter 2011). For instance, the equation-oriented Modelica language (Elmqvist, Mattsson, and Otter 1999; Fritzson and Engelson 1998) allows modelers to express detailed multi-physics descriptions of thermo-fluid systems (Laughman 2014). Other tools, such as EnergyPlus, DOE-2, ESP-r, TRNSYS have all been compared in the literature (Sousa 2012; Wetter, Treeck, and Hensen 2013).

These models are often coupled and run concurrently to make use of results generated by other models at runtime (Nicolai and Paepcke 2017). For example, a building energy simulation model computing room air temperatures may require heating loads from an HVAC supply system, with the latter coming from a simulation model external to the building simulation tool. Thus, integration of these models into a common interface to make use of their different features, while challenging (Wetter, Treeck, and Hensen 2013), is an important task.

While the above challenge has been addressed by FMI, the resulting coupled simulation using FMUs is computationally expensive due to the underlying numerical stiffness (Robertson and Williams 1975) widely prevalent in many engineering models. These simulations require adaptive implicit integrators to step forward in time (Wanner and Hairer 1996). For example, building heat transfer dynamics has time constants in hours, whereas feedback controllers have time constants in seconds. Thus, surrogate models are often used in building simulation (Westermann and Evins 2019).

In the following sections, we describe surrogate generation of a complex Room Air Conditioner (RAC) model, which has been exported as an FMU. We then use the surrogate to find the optimal set of design parameters over which system performance is maximized, yielding two orders of magnitude speedup over using the full model. Finally, we discuss the deployment of the surrogate in a co-simulation loop coupled with another FMU.

3.1 Surrogates of Coupled RAC Models

We first consider surrogate generation of a Room Air Conditioner (RAC) model using JuliaSim, consisting of a coupled room model with a vapor compression cycle model, which removes heat from the room and dissipates it outside. This model was provided to us by a user as-is, and a maximum relative error tolerance of 5% was chosen. The vapor compression cycle itself consists of detailed physics-based component models of a compressor, an expansive valve and a finite volume, and a staggered-grid dynamic heat exchanger model (Laughman 2014). This equipment is run open-loop in this model to simplify the interactions between the equipment and the thermal zone. The room model is designed using components from the Modelica Buildings library (Wetter, Zuo, et al. 2014). The room is modeled as a volume of air with internal convective heat gain and heat conduction outside. The Chicago O'Hare TMY3 weather dataset[9] is imported and is used to define the ambient temperature of the air outside. This coupled model is written and exported from Dymola 2020x as a co-simulation FMU.

The model is simulated with 100 sets of parameters sampled from a chosen parameter space using Latin hypercube sampling. The simulation timespan was a full day with a fixed step size of 5 seconds. The JuliaSim FMU simulation backend runs simulations for each parameter set in parallel and fits cubic splines to the resulting time series outputs to continuously sample points from parts of the trajectory. Then the CTESN algorithm computes projections from the reservoir time series to output time series at each parameter set. Finally, a radial basis function creates a nonlinear map between the chosen parameter space and the space of projections. Figure 2 and Table 2 show the relative errors when the surrogate is tested at a parameter set on which it has not been trained. To demonstrate the reliability of the surrogate through the chosen parameter space, 100 further test parameters were sampled from the space, and the errors for each test were compiled into a histogram, as shown in 3. At any test point, the surrogate takes about 6.1 seconds to run, while the full model takes 35 minutes, resulting in a speedup of 344x.

This surrogate model can then be reliably deployed for design and optimization, which is outlined in the following section.

3.2 Accelerating Global Optimization

Building design optimization (Nguyen, Reiter, and Rigo 2014; Machairas, Tsangrassoulis, and Axarli 2014) has benefited from the use of surrogates by accelerating optimization through faster function evaluations and smooth-

[9] https://bcl.nrel.gov/node/58958

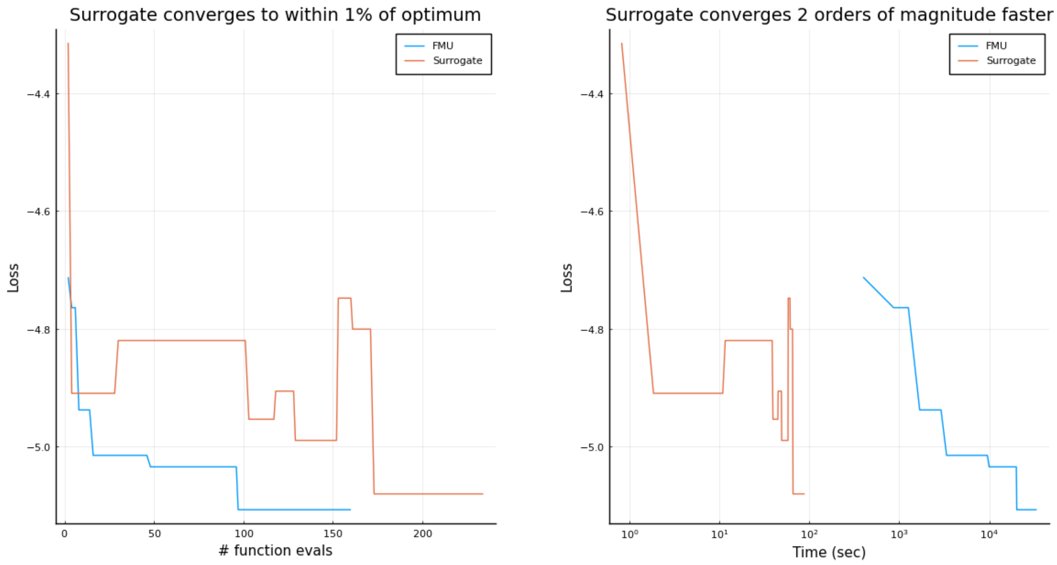


Figure 4. Comparison of global optimization while using the full model and the surrogate. Loss is measured using the full model's objective function. (Left) Convergence of loss with number of function evaluations (Right) Convergence of loss with wall clock time. The optimization using the surrogate converged much before the result from the first function evaluation of the full model is over. This is why the blue line appears translated horizontally in time.

ing objective functions with discontinuities (Westermann and Evins 2019; Wetter and Wright 2004).

The quantity to be maximized (or whose negative value is to be minimized) is the average coefficient of performance (COP) across the time span. We calculate this using output time series from the model by means of the following formula:

$$COP(t) = \frac{Q_{tot}(t)}{\max(0.01, CSP(t))} \tag{11}$$

$$COP_{avg} = \frac{\sum_{n=1}^{N_t} COP(t_n)}{N_t} \tag{12}$$

where COP refers to the coefficient of performance, COP_{avg} refers to the average coefficient of performance across the time interval (the quantity to optimize), Q_{tot} the total heat dissipation from the coupled model, $CSP(t)$ is the compressor shaft power, and N_t represents the number of points in time sampled from the interval (720).

We use an adaptive differential evolution global optimization algorithm, which does not require the calculation of gradients or Hessians (Price, Storn, and Lampinen 2006). We chose this algorithm because of its ability to handle black-box objective functions. We use the differential optimizers in BlackBoxOptim.jl[10] for this experiment.

Figure 4 shows that the surrogate produces a series of minimizers, which eventually converge to within 1% of the reference minimum value chosen, but two orders of magnitude faster. The surrogate does take more function evaluations to converge than the true model, but since each function value is relatively inexpensive, the impact on wall clock time is negligible.

3.3 Co-simulation with Surrogates

Next we examine a co-simulation loop with two coupled FMUs and replace one of the FMUs with a surrogate. Co-simulation is a form of coupled simulation where a master algorithm simulates and synchronizes time dependent models models at discrete time steps. An advantage of co-simulation over model exchange is that the individual FMUs can be shipped with their own solvers. These FMU solver calls are abstracted away from the master algorithm, which only pays heed to initialization and synchronization of the FMUs.

We examine a simplified example of an HVAC system providing cooling to a room from the Modelica Buildings library (Wetter, Bonvini, et al. 2015). Both the HVAC system and room models have been exported as FMUs, which are then imported into JuliaSim and then coupled via co-simulation. At each step of the co-simulation, the models are simulated for a fixed time step, and the values of the coupling variables are queried and then set as inputs to each other, before the models are simulated at the next time step.

JuliaSim then generates a surrogate of the HVAC system by training over the set of inputs received during the co-simulation loop. It is then deployed in a "plug and play" fashion, by coupling the outputs of the surrogates to the inputs of the room and vice versa. The resultant output from the coupled system is shown in Figure 5. The above co-simulation test has been conducted at the same set of set of design parameters as the original simulation. [11] While the individual models in this test are simplified,

[10] https://github.com/robertfeldt/BlackBoxOptim.jl

[11] We tried to simulate this coupled system at different design parameters, but were unable to, for reasons currently unknown to us, change certain parameters on Dymola 2020x. We were also not able

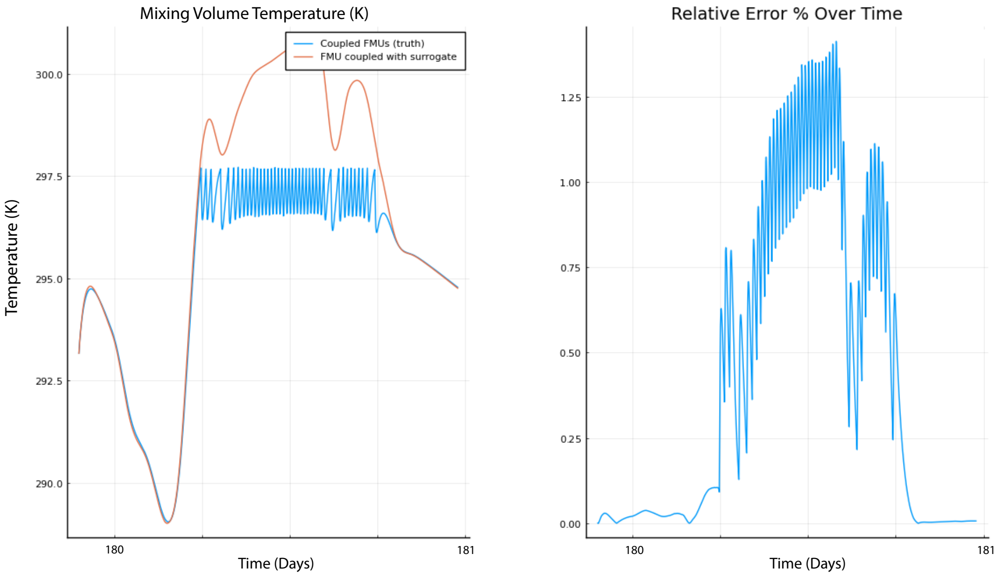


Figure 5. Coupled co-simulation of a surrogate and an FMU. The blue line represents the ground truth, which is the output from the co-simulation of two coupled FMUs, and the red line represents the output from the coupled surrogate and an FMU. While the prediction smooths over transients found in the ground truth, it does so at a relative error of less than 1.5%. This result also empirically suggests that the output from the surrogate is bounded over the set of inputs it has received over co-simulation. The surrogate was trained over a sample of 100 inputs received from the room model. The error over the transients can be reduced by sampling more inputs from the co-simulation.

they serve as a proof of concept for a larger coupled simulation, either involving more FMUs or involving larger models, which may be prohibitively expensive (Wetter, Fuchs, and Nouidui 2015).

4 Conclusion

We demonstrate the capabilities of JuliaSim, a software for automated generation of deployment of surrogates for design, optimization and coupled simulation. Our surrogates can reproduce outputs from detailed multi-physics systems and can be used as stand-ins for global optimization and coupled simulation. Our results show the promise of blending machine learning surrogates in JuliaSim, and we believe that it can enable a machine learning-accelerated workflow for design and development of complex multi-physical systems.

There are many avenues for this work to continue. Further work to deploy these embedded surrogates as FMUs themselves is underway. This would allow JuliaSim to ship accelerated FMUs to other platforms. Other surrogate algorithms, such as proper orthogonal decomposition (Chatterjee 2000), neural ordinary differential equations (Chen et al. 2018; S. Kim et al. 2021), and dynamic mode decomposition (Schmid 2010) will be added in upcoming releases and rigorously tested on the full model library. Incorporating machine learning in other fashions, such as within symbolic simplification algorithms, is similarly being explored. But together, JuliaSim demonstrates that future modeling and simulation software does not need to, and should not, eschew all of the knowledge of the past equation-based systems in order to bring machine learning into the system.

Acknowledgements

The information, data, or work presented herein was funded in part by ARPA-E under award numbers DE-AR0001222 and DE-AR0001211, and NSF award number IIP-1938400. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

References

Anantharaman, Ranjan et al. (2021). “Accelerating Simulation of Stiff Nonlinear Systems using Continuous-Time Echo State Networks”. In: *Proceedings of the AAAI 2021 Spring Symposium on Combining Artificial Intelligence and Machine Learning with Physical Sciences*.

Benner, Peter, Serkan Gugercin, and Karen Willcox (2015). “A survey of projection-based model reduction methods for parametric dynamical systems”. In: *SIAM review* 57.4, pp. 483–531.

Bezanson, Jeff et al. (2017). “Julia: A fresh approach to numerical computing”. In: *SIAM review* 59.1, pp. 65–98.

Blochwitz, Torsten et al. (2011). “The functional mockup interface for tool independent exchange of simulation models”. In: *Proceedings of the 8th International Modelica Conference*. Linköping University Press, pp. 105–114.

to change those parameters having exported the constituent models as co-simulation FMUs. We shall aim to debug this issue and complete this story in future work.

Brück, Dag et al. (2002). "Dymola for multi-engineering modeling and simulation". In: *Proceedings of modelica*. Vol. 2002. Citeseer.

Chatterjee, Anindya (2000). "An introduction to the proper orthogonal decomposition". In: *Current science*, pp. 808–817.

Chen, Ricky TQ et al. (2018). "Neural ordinary differential equations". In: *arXiv preprint arXiv:1806.07366*.

Cohen, Scott D, Alan C Hindmarsh, and Paul F Dubois (1996). "CVODE, a stiff/nonstiff ODE solver in C". In: *Computers in physics* 10.2, pp. 138–143.

Elmqvist, Hilding, Sven Erik Mattsson, and Martin Otter (1999). "Modelica-a language for physical system modeling, visualization and interaction". In: *Proceedings of the 1999 IEEE international symposium on computer aided control system design (Cat. No. 99TH8404)*. IEEE, pp. 630–639.

Fritzson, Peter, Peter Aronsson, et al. (2005). "The OpenModelica modeling, simulation, and development environment". In: *46th Conference on Simulation and Modelling of the Scandinavian Simulation Society (SIMS2005), Trondheim, Norway, October 13-14, 2005*.

Fritzson, Peter and Vadim Engelson (1998). "Modelica—A unified object-oriented language for system modeling and simulation". In: *European Conference on Object-Oriented Programming*. Springer, pp. 67–90.

Gobbert, Matthias K (1996). "Robertson's example for stiff differential equations". In: *Arizona State University, Technical report*.

Hosea, ME and LF Shampine (1996). "Analysis and implementation of TR-BDF2". In: *Applied Numerical Mathematics* 20.1-2, pp. 21–37.

Hu, Liwei et al. (2020). "Neural networks-based aerodynamic data modeling: A comprehensive review". In: *IEEE Access* 8, pp. 90805–90823.

Kim, Suyong et al. (2021). *Stiff Neural Ordinary Differential Equations*. arXiv: 2103.15341 [math.NA].

Kim, Youngkyu et al. (2020). "A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder". In: *arXiv preprint arXiv:2009.11990*.

Laughman, Christopher R (2014). "A Comparison of Transient Heat-Pump Cycle Simulations with Homogeneous and Heterogeneous Flow Models". In:

Lukoševičius, Mantas (2012). "A practical guide to applying echo state networks". In: *Neural networks: Tricks of the trade*. Springer, pp. 659–686.

Lukoševičius, Mantas and Herbert Jaeger (2009). "Reservoir computing approaches to recurrent neural network training". In: *Computer Science Review* 3.3, pp. 127–149.

Ma, Yingbo et al. (2021). "ModelingToolkit: A Composable Graph Transformation System For Equation-Based Modeling". In: *arXiv preprint arXiv:2103.05244*.

Machairas, Vasileios, Aris Tsangrassoulis, and Kleo Axarli (2014). "Algorithms for optimization of building design: A review". In: *Renewable and sustainable energy reviews* 31, pp. 101–112.

Nguyen, Anh-Tuan, Sigrid Reiter, and Philippe Rigo (2014). "A review on simulation-based optimization methods applied to building performance analysis". In: *Applied Energy* 113, pp. 1043–1058.

Nicolai, Andreas and Anne Paepcke (2017). "Co-Simulation between detailed building energy performance simulation and Modelica HVAC component models". In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. 132. Linköping University Electronic Press, pp. 63–72.

Otter, Martin and Hilding Elmqvist (2017). "Transformation of differential algebraic array equations to index one form". In: *Proceedings of the 12th International Modelica Conference*. Linköping University Electronic Press.

Pantelides, Constantinos C. (1988). "The Consistent Initialization of Differential-Algebraic Systems". In: *SIAM Journal on Scientific and Statistical Computing* 9.2, pp. 213–231. DOI: 10.1137/0909014.

Price, Kenneth, Rainer M Storn, and Jouni A Lampinen (2006). *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media.

Rackauckas, Christopher and Qing Nie (2017). "Differentialequations. jl–a performant and feature-rich ecosystem for solving differential equations in julia". In: *Journal of Open Research Software* 5.1.

Raissi, Maziar, Paris Perdikaris, and George E Karniadakis (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378, pp. 686–707.

Ratnaswamy, Vishagan et al. (2019). "Physics-informed Recurrent Neural Network Surrogates for E3SM Land Model". In: *AGU Fall Meeting Abstracts*. Vol. 2019, GC43D–1365.

Robertson, HH (1976). "Numerical integration of systems of stiff ordinary differential equations with special structure". In: *IMA Journal of Applied Mathematics* 18.2, pp. 249–263.

Robertson, HH and J Williams (1975). "Some properties of algorithms for stiff differential equations". In: *IMA Journal of Applied Mathematics* 16.1, pp. 23–34.

Schmid, Peter J (2010). "Dynamic mode decomposition of numerical and experimental data". In: *Journal of fluid mechanics* 656, pp. 5–28.

Sousa, Joana (2012). "Energy simulation software for buildings: review and comparison". In: *International Workshop on Information Technology for Energy Applicatons-IT4Energy, Lisabon*.

Wang, Sifan, Yujun Teng, and Paris Perdikaris (2020). "Understanding and mitigating gradient pathologies in physics-informed neural networks". In: *arXiv preprint arXiv:2001.04536*.

Wanner, Gerhard and Ernst Hairer (1996). *Solving ordinary differential equations II*. Vol. 375. Springer Berlin Heidelberg.

Westermann, Paul and Ralph Evins (2019). "Surrogate modelling for sustainable building design–A review". In: *Energy and Buildings* 198, pp. 170–186.

Wetter, Michael (2011). *A view on future building system modeling and simulation*. Tech. rep. Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).

Wetter, Michael, Marco Bonvini, et al. (2015). "Modelica buildings library 2.0". In: *Proc. of The 14th International Conference of the International Building Performance Simulation Association (Building Simulation 2015), Hyderabad, India*.

Wetter, Michael, Marcus Fuchs, and Thierry Nouidui (2015). "Design choices for thermofluid flow components and systems that are exported as Functional Mockup Units". In:

Wetter, Michael, Christoph van Treeck, and Jan Hensen (2013). "New generation computational tools for building and community energy systems". In: *IEA EBC Annex* 60.

Wetter, Michael and Jonathan Wright (2004). "A comparison of deterministic and probabilistic optimization algorithms for nonsmooth simulation-based optimization". In: *Building and Environment* 39.8, pp. 989–999.

Wetter, Michael, Wangda Zuo, et al. (2014). "Modelica buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270.

Willard, Jared et al. (2020). "Integrating physics-based modeling with machine learning: A survey". In: *arXiv preprint arXiv:2003.04919*.

Yu, Tommy et al. (2011). "The physiome model repository 2". In: *Bioinformatics* 27.5, pp. 743–744.

Zhang, Ruixi et al. (2020). "Hydrological Process Surrogate Modelling and Simulation with Neural Networks". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, pp. 449–461.

OpenModelica.jl: A modular and extensible Modelica compiler framework in Julia targeting ModelingToolkit.jl

John Tinnerholm[1] Adrian Pop[1] Andreas Heuermann[2] Martin Sjölund[1]

[1]Department of Computer and Information Science, Linköping University, Sweden, {first.last}@liu.se
[2]Faculty of Engineering and Mathematics, Bielefeld University of Applied Sciences, Germany,
{first.last}@fh-bielefeld.de

Abstract

This paper presents current work on our Modelica Compiler framework in Julia: OpenModelica.jl.[1] We provide a brief overview of this novel framework and its features, and we also present the latest addition to the possible backend options. We target ModelingToolkit.jl (MTK), a framework for symbolic-numerical computation and scientific machine learning. We evaluated the performance of our new backend using the ScalableTestsuite, a benchmark suite for Modelica Compilers. In our experiment, we demonstrate that MTK can be used as a backend with competitive simulation performance. In addition, using the scientific machine learning features of the Modeling toolkit, we were able to approximate models in the ScalableTestsuite using surrogate techniques and how such techniques can be used to accelerate the solving of nonlinear algebraic loops during tearing.

Based on our experiments, we propose using this new framework to automatically generate surrogate components of a Modelica model during the simulation to increase performance. The experimental work presented here provides one of the first investigations concerning the integration of the symbolic-numerical abilities of Julia within a Modelica tool.

Keywords: Modelica, OpenModelica, Julia, Equation-based modeling, Compiler-construction

1 Introduction

The ability to model cyber-physical systems (CPS) is essential for many scientific and industrial processes. Modelica is a standardized declarative equation-based object-oriented language with a solid tool and library support. Recently, researchers have shown an increased interest in the Julia language (Bezanson et al. 2017), with the release of several packages that bring acausal modeling to Julia, such as Modia (Elmqvist and Otter 2017) and ModelingToolkit (MTK) (Ma et al. 2021).

Thus several studies have begun to examine the implications of using Julia as a foundation to design new modeling frameworks. In this paper, we present our contribution to this effort within the OpenModelica programming environment (Fritzson, Pop, Abdelhak, et al. 2020).

1.1 Motivation

The main motivation for the work presented here is that previous studies do not attempt to integrate Modelica within Julia. Instead, they provide the possibility of Modelica-like acausal modeling using Julia as a host language. Tinnerholm et al. (2020) presented our first Modelica compiler prototype in Julia. This compiler was developed with the goal to utilize Julia's symbolic-numerical capabilities and extend the current capabilities of Modelica. In this paper, we expand this work to implement a full Modelica compiler using Julia with the goal to improve and optimize existing models, and by adhering to the standards of the Modelica language, we hope to facilitate the reuse of modeling know-how contained in existing Modelica libraries. Updates to this first iteration of this compiler include automatic translation of the high-performance frontend (Pop et al. 2019) along with experimental support for hybrid systems and a new backend targeting ModelingToolkit. We have used this framework to simulate Modelica models of systems containing thousands of equations and variables to assess the performance of our compiler.

1.2 Contributions

While several Modelica Compilers have been designed before, no compiler has previously used Julia as an implementation language. Thus, a central contribution of this paper is evidence that such a compiler is both feasible and is easily extendable. We demonstrate this by using the symbolic-numerical capabilities of Julia and the scientific machine learning capabilities of MTK to automatically generate surrogate models within a modular and extendable pipeline. Another contribution of this paper is, to our knowledge, the first empirical investigation concerning the performance characteristics of ModelingToolkit when employed as a backend for a Modelica Compiler, demonstrating its claimed usefulness as a compiler component for equation-oriented languages.

1.3 Paper Organization

This paper is organized as follows: The background is presented in section 2, this is followed by section 3 where we present the structure of the compiler. In section 4 we recount how we verified the frontend together with some

[1] On GitHub: OpenModelica/OpenModelica.jl

DOI
10.3384/ecp21181109

current performance characteristics when flattening the cascaded first-order system from the ScalableTestsuite in Listing 4 along with a description of experiments to assess the correctness of the frontend. This section is followed by section 5 which provides a benchmark highlighting the current simulation performance of the new MTK backend and comparing it to the OpenModelica Compiler (OMC). This is followed by section 6 in which we demonstrate how the scientific machine learning features of MTK can be used to approximate models or subcomponents of models. In section 7 where we present our conclusions along with recommendations for future work.

2 Background

As stated in section 1 there exist as of 2021 several modeling environments that provide the option of causal and acausal modeling within the Julia ecosystem. *DifferentialEquations.jl* (Rackauckas and Nie 2017) is one such environment. It provides a seamless foreign function interface that allows interfacing algorithmic Julia code and a variety of different solvers. A user of DifferentialEquations.jl writes imperative code in the Julia language to conform to systems such as Nonlinear-systems, ODE-systems, and DAE-systems. Tinnerholm et al. (2020) selected DifferentialEquations.jl as the default backend target. A model of a hybrid system representing a bouncing ball using DifferentialEquations.jl can be studied in Listing 1.

While DifferentialEquations.jl provides the abstractions necessary to write causal models in Julia, it does not provide the abstractions of a full-fledged modeling language. *ModelingToolkit.jl* (MTK) aims address this issue (Ma et al. 2021). MTK is a recent modeling framework to automate symbolic operations common for equation-oriented languages, such as methods for index reduction. It does so by using the symbolic-numerical capabilities of Julia to preprocess an MTK model description into a format that can be solved using the set of solvers provided by DifferentialEquations.jl. In other words, the iterative process from an acausal description based on equations to a causal representation acceptable for a solver is similar to that of a typical Modelica Compiler. The language defined by MTK does not at the time of this writing support hybrid systems. However, it is possible to post-process MTK models to add events similar to Listing 1 where the Modelica *when-equation* is represented using a `ContinuousCallback` which is illustrated in Listing 2. If we compare the generated code in Listing 1 with that of Listing 2 we can see that MTK is closer to Modelica in the level of abstraction; however, MTK lacks control structures found in Modelica such as **`for`** and **`if`**.

MTK does not only target DAE-systems, it also targets several areas which are not the primary target of the Modelica language such as:

- Stochastic differential(-algebraic) equations
- Partial differential equations
- Optimization problems
- Continuous-Time Markov Chains
- Nonlinear Optimal Control

This enables users of MTK to combine different systems from different domains (Ma et al. 2021). Conceivably, model exchange between this framework and Modelica would be useful for efficient modeling and simulation of large dynamic systems.

The main difference between Modelica and the language defined by MTK is the level of abstraction. To give an example, as of this writing, ModelingToolkit.jl requires users to specify the application of index reduction explicitly; it also requires systems to be specified explicitly with the state derivatives on the right-hand side. Thus, the user specifies the transformation from a DAE-System into an ODE-system, whereas in a Modelica compiler, these decisions are generally abstracted away. Still, as we will illustrate in this paper, MTK is suitable as a backend framework for Modelica Compilers or other equation-oriented languages frameworks in Julia.

Modia.jl (Elmqvist and Otter 2017) is another framework that brings acausal modeling to Julia. Syntactically it is more similar to Modelica when compared to the language defined by MTK. However, it is different from the work presented here because its constructs are implemented using Julia metaprogramming rather than traditional data structures used by compilers.

Yet another modeling framework is *Causal.jl* (Sarı and Günel 2019). However, as the name implies, it is a causal modeling framework reminiscent of Simulink.

3 Compiler Structure

In this section, we will elaborate on the different components that make up OpenModelica.jl. To provide a brief overview of the size of this application, a summary of the current size of this compiler by lines of code (LOC) is provided in Table 1. For comparison, the OMC compiler has about 1,100,000 LOC MetaModelica code for the frontend+backend and about 67,000 LOC C code for the runtime system. Using Julia is clearly an advantage as one can delegate functionalities such as finding strongly connected components to existing Julia libraries.

The frontend is made up of OMParser and OMFrontend; the backend of OMBackend and the runtime of MetaModelica.jl. Internally three intermediate representations are used: Absyn[2], SCode[3] and DAE.[4] An overview of the compiler pipeline is presented in Figure 1. An example of how to simulate and plot a Modelica model in Julia is given in Listing 3.

[2] On GitHub: OpenModelica/Absyn.jl

[3] On GitHub: OpenModelica/SCode.jl

[4] On GitHub: OpenModelica/DAE.jl

Listing 1. Automatically generated Julia code for a simple hybrid system. The Julia code presented in this listing is targeting the IDA solver in Sundials (Hindmarsh et al. 2005) using DifferentialEquations.jl.

```
function BouncingBallRealsStartConditions(
    aux, t)
  local x = zeros(2)
  local dx = zeros(2)
  local p = aux[1]
  local reals = aux[2]
  reals[1] = 1.0
  dx[1] = reals[2]
  dx[2] = -(p[2])
  x[2] = reals[2]
  x[1] = reals[1]
  return (x, dx)
end
function BouncingBallRealsDifferentialVars
    ()
  return Bool[1, 1]
end
function BouncingBallRealsDAE_equations(res
    , dx, x, aux, t)
  local p = aux[1]
  local reals = aux[2]
  res[1] = dx[2] - -(p[2])
  res[2] = dx[1] - reals[2]
  reals[2] = x[2]
  reals[1] = x[1]
end
function BouncingBallRealsParameterVars()
  local aux = Array{Array{Float64}}(undef,
      2)
  local p = Array{Float64}(undef, 2)
  local reals = Array{Float64}(undef, 2)
  aux[1] = p
  aux[2] = reals
  p[2] = 9.81
  p[1] = 0.7
  return aux
end
saved_values_BouncingBallReals =
    SavedValues(Float64, Tuple{Float64,
    Array})
function BouncingBallRealsCallbackSet(aux)
  local p = aux[1]
  function condition1(x, t, integrator)
    x[1] - 0.0
  end
  function affect1!(integrator)
    integrator.u[2] = -(p[1] * integrator.u
        [2])
  end
  cb1 = ContinuousCallback(
         condition1,
          affect1!,
          rootfind = true,
          save_positions = (true, true),
          affect_neg! = affect1!,
        )
  savingFunction(u, t, integrator) =
    let
      (t, deepcopy(integrator.p[2]))
     end
  cb2 = SavingCallback(savingFunction,
      saved_values_BouncingBallReals)
  return CallbackSet(cb1, cb2)
end
```

Listing 2. An MTK version of the bouncing ball produced by the new backend.

```
using ModelingToolkit
using DiffEqBase
using DifferentialEquations
function BouncingBallRealsModel(tspan =
    (0.0, 1.0))
  pars = ModelingToolkit.@parameters(begin
      (e, g, t) end)
  vars = ModelingToolkit.@variables(begin (
      h(t), v(t)) end)
  der = Differential(t)
  eqs = [
    der(h) ~ v,
    der(v) ~ -g
  ]
  nonLinearSystem = ModelingToolkit.
      ODESystem(eqs, t, vars, pars,
    name = :($(Symbol("BouncingBallReals"))
        ),
  )
  pars = Dict(e => float(0.7), g => float
      (9.81), t => tspan[1])
  initialValues = [h => 1.0, v => 0.0]
  firstOrderSystem = ModelingToolkit.
      ode_order_lowering(nonLinearSystem)
  reducedSystem = ModelingToolkit.
      dae_index_lowering(firstOrderSystem)
  problem = ModelingToolkit.ODEProblem(
      reducedSystem, initialValues, tspan,
      pars)
  return problem
end
function BouncingBallRealsCallbackSet()
  function condition1(x, t, integrator)
    x[1] - 0.0
  end
  function affect1!(integrator)
    integrator.u[2] = -(integrator.p[1] *
        integrator.u[2])
  end
  cb1 = ContinuousCallback(
    condition1,
    affect1!,
    rootfind = true,
    save_positions = (true, true),
    affect_neg! = affect1!,
  )
  return CallbackSet(cb1)
end
BouncingBallRealsModel_problem =
    BouncingBallRealsModel()
function BouncingBallRealsSimulate(tspan =
    (0.0, 1.0))
  solve(BouncingBallRealsModel_problem,
      tspan = tspan, callback =
      BouncingBallRealsCallbackSet())
end
function BouncingBallRealsSimulate(tspan =
    (0.0, 1.0); solver = Tsit5())
  solve(BouncingBallRealsModel_problem,
      tspan = tspan, solver)
end
```

Listing 3. This listing illustrates how to export and simulate a Modelica model in Julia. Once preprocessed by the frontend the call to *OMBackend.translate* stores the final MTK program for future analysis or simulation. The input parameter modelName provided by the caller and it is assumed that the file containing a Modelica model is named after this parameter.

```
function runTest(modelName)
  #= OMParser phase =#
  ast::Absyn.Program = OMFrontend.parseFile
      ("./Models/$(modelName).mo")
  #= OMFrontend phases =#
  scodeProgram::SCode.Program = OMFrontend.
      translateToSCode(ast)
  (dae, cache) = OMFrontend.
      instantiateSCodeToDAE(modelName,
      scodeProgram)
  #= Backend phases =#
  OMBackend.translate(dae; BackendMode =
      OMBackend.MODELING_TOOLKIT_MODE)
  res = OMBackend.simulateModel(modelName,
      tspan = (0.0, 1.0))
  #= Optionally plot the result =#
  OMBackend.plot(res)
end
```

Table 1. Current sizes of OpenModelica.jl compiler phases by LOC.

Compiler Phase	*Lines*
Runtime	1,853
FrontEnd	135,103
BackEnd & Code generation	6,073
Total size	*143,029*

3.1 OMParser.jl

The parser, OpenModelicaParser.jl was adapted from the existing OMC parser.[5] which is written in ANTLR (Parr and Quong 1995) It is currently capable of parsing large files such as the entire Modelica Standard Library.[6] After a successful parse, the AST can be fed to the frontend module, *OMFrontend.jl* the data structure representing the AST is defined by *Absyn.jl*.[2] In Listing 3 this parser is invoked by `OMFrontend.parseFile("./Models/$(modelName).mo")`.

3.2 OMFrontend.jl

To flatten the Modelica code, we use the OMFrontend.jl, which was automatically generated from the high-performance frontend of the OMC (Pop et al. 2019). Previously, we used the old frontend (Tinnerholm et al. 2020); however, as part of the work presented here, the *MetaModelica-Julia translator* introduced by Tinnerholm et al. (2020) was used to automatically generate a Julia implementation of the high-performance frontend (Pop et al. 2019). Consequently, the frontend is implemented in automatically generated Julia code generated by translating the existing OpenModelica frontend. While the translation of the old frontend[7] was achieved without any major modifications, we had to manually resolve cases of mutually circular module dependencies for the new since Julia does not handle them while MetaModelica does.

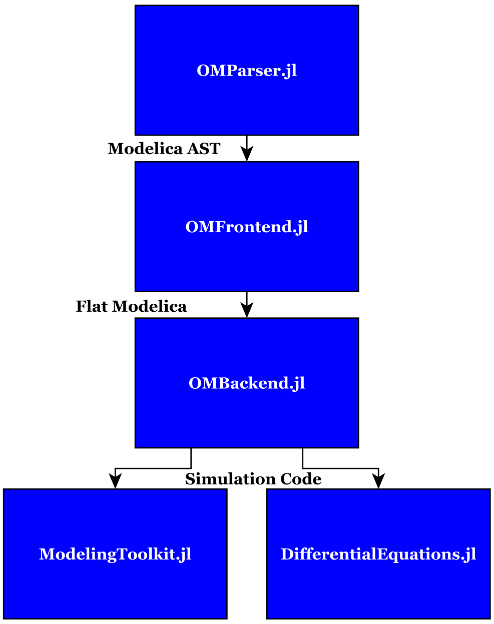


Figure 1. An illustration of the compiler pipeline including current available backends targeting ModelingToolkit.jl and DifferentialEquations.jl. The Modelica AST is represented using Absyn.jl. Inside OMFrontend.jl the SCode representation defined by SCode.jl is used. Flat Modelica is encoded using DAE.jl

3.3 OMBackend.jl

OMBackend is the backend module of this compiler, and it is implemented as a separate package. Current backend targets include both ModelingToolkit and DifferentialEquations.jl. Simulations targeting DifferentialEquations.jl use the Sundials IDA solver and is based on the DAE-mode implementation by W. Braun, Casella, Bachmann, et al. (2017). It currently provides support for continuous systems and experimental support for hybrid systems. An example of code generated for a hybrid system is the bouncing ball model, see Listing 1.

3.3.1 The MTK-Backend

The new backend based on ModelingToolkit (MTK-Backend) is capable of automatically translating Modelica models into equivalent MTK models. The MTK backend works by accepting the flat Modelica/Hybrid DAE that is described by *DAE.jl*. Since MTK automatically handles transformations such as index reduction, no such algorithm is applied by the backend. MTK is also acausal in

[5] On GitHub: adrpo/OpenModelicaParser.jl

[6] Modelica Standard Library (Version 3.2.1) On GitHub: modelica/ModelicaStandardLibrary

[7] The old frontend is the frontend the *high-performance frontend* replaced (Pop et al. 2019).

the sense that there need not to be a causal order between the equations. However, MTK requires the system to be in explicit form, so the derivatives are reordered using MTK's symbolic algebra routines. An MTK translation of the Modelica code in Listing 4 can be studied in Listing 5. In Listing 5 the function Casc10Model defines the model, the parameters are defined using *parameters* and the variables are defined using *variables*. The statement *ode_order_lowering* transforms the system to first-order form and *dae_index_lowering* performs index reduction[8].

3.4 MetaModelica.jl

MetaModelica.jl[9] provides a compatibility layer between Julia and MetaModelica (Fritzson, Pop, and Aronsson 2005). It reimplements several constructs of MetaModelica such as *match* and *matchcontinue*. Furthermore, MetaModelica.jl replicates the existing runtime of OMC. This package is used extensively in the translated modules.

4 Verification

The compiler presented in this paper consists of several components, see Figure 1 where each component contains thousands of LOC, see Table 1.

To verify the parser, we parsed the Modelica Standard Library, along with some other models, some of which contained errors. This was done to establish that it reported the same errors as the existing parser in OpenModelica.[10] Verifying the correctness of the frontend was more difficult since it consists of over 130,000 lines of automatically generated Julia code. We tested our implementation by lowering the hybrid DAE produced by the frontend and compared the result of simulating these models with the simulation results of OpenModelica. One excerpt of the verification experiments can be studied in Figure 2. The runnable code is available in listing 5. The corresponding Modelica model is available in Listing 4. The results are by no means exhaustive, but our preliminary verification experiments suggest that the translated frontend behaves correctly for the set of models that we tested.

5 Simulation Performance

The previous section has shown that we can generate and simulate Modelica models targeting DifferentialEquations.jl and MTK directly. In this section, we present the current simulation performance of the new MTK backend using the cascaded first-order system from the ScalableTestsuite (Casella 2015) and how the simulation performance of the MTK backend compares to OpenModelica. Two model from the benchmark suite where used *CascadedFirstOrder*, see Listing 4. The parameters of these were modified to increase the number of equations and events gradually.

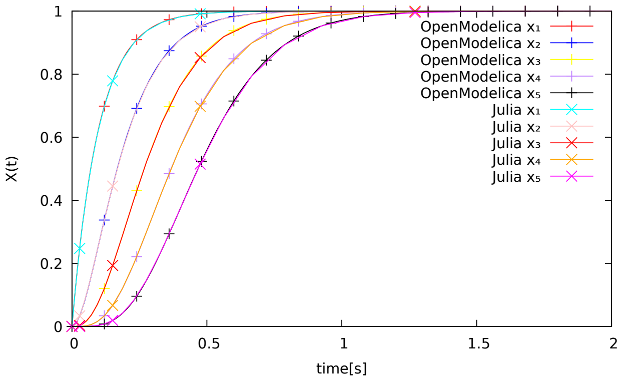


Figure 2. The result of simulating the cascading first-order system between 0.0 to 2.0 seconds, with $N = 10$ simulated using the new MTK backend and OMC. The plot shows $x_1, ..., x_5$ of $X(t)$. Since the curves are almost identical, markers are added to differentiate between the MTK and OpenModelica solutions.

5.1 Experimental setup

In this subsection, we describe the experimental setup of the model included our experiments.

To assess simulation performance of *CascadedFirstOrder* we generated code using the model in Listing 4 with the following values for the parameter N: $10, 100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600$. The resulting model with $N = 10$ can be studied in Listing 5. However, due to long compilation times of MTK when $N > 3200$ we limited[11] our benchmark to the following values for N: $10, 100, 200, 400, 800, 1600, 3200$. We simulated models using ModelingToolkit v5.16.0 with Julia 1.6.1 and OpenModelica 1.18.0. All tests were run on a 16-core AMD TR 1950X with 128GB of RAM. When measuring simulation times for OMC and for Julia, we used Benchmarktools.jl (Chen and Revels 2016) Two experiments were run for the *CascadedFirstOrder* model.

For the first experiment we used the following solvers:

- **OMC** Sundials IDA solver (Hindmarsh et al. 2005)
- **MTK** The default solver,[12] invoked when calling solve without auxiliary arguments

In the second experiment, we selected Tsit5 (Tsitouras 2011) as the solver for MTK. For OpenModelica, we kept the IDA solver since, at the time of writing, the Tsit5-solver is not available within the OpenModelica environment.

5.2 Evaluating simulation performance

In Figure 3 we present the result of our first experiment. From the graph, we can see that the simulation time of

[8] Index reduction and the lowering of the ODE is not always necessary. For instance, an index 1 DAE does not need to have its' index reduced. Still, for generality, we do this for all systems.

[9] On GitHub: OpenModelica/MetaModelica.jl

[10] A subset of these tests are available on GitHub: adrpo/OpenModelicaParser.jl/test

[11] The MTK-models where $N > 3200$ are available on request.

[12] The default solver is selected automatically by DifferentialEquations.jl depending on the characteristics of the model (Rackauckas and Nie 2019). See solver selection algorithm.

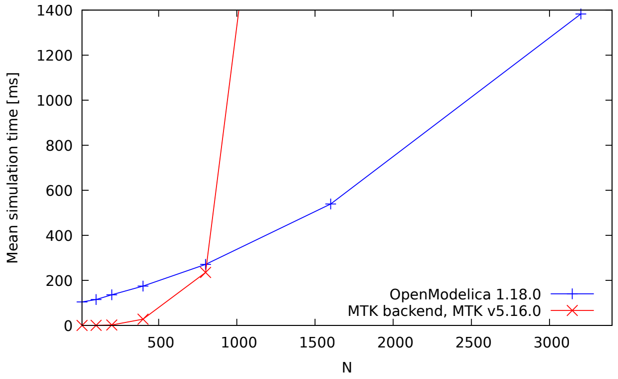


Figure 3. The mean simulation time of the cascading first-order system, with $N = 10, 100, 200, 400, 800, 1600, 3200$ simulated using the new MTK backend and OMC.

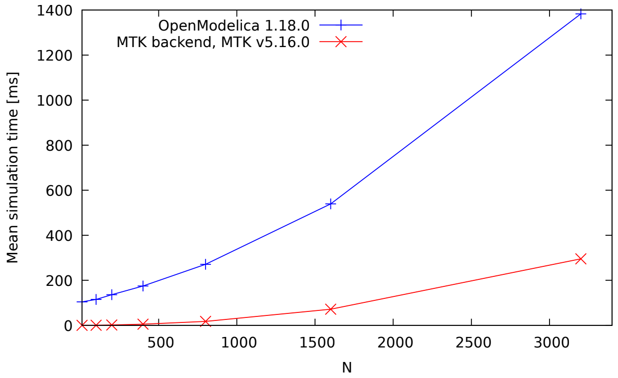


Figure 4. Same simulation as in Figure 3 using Tsit5 together with MTK.

MTK is superior to the OMC until $N > 400$, then the simulation time of MTK increases exponentially. This is probably due to conservative solver selection by MTK. This behavior was also observed when we compiled the automatically generated MTK code, where we observed an exponential increase in compilation time when $N > 3200$. Because of this, we omitted these models from the experiments. The result of the second experiment can be studied in Figure 4, it demonstrates superior simulation performance of MTK when compared to OMC when using an alternative solver, Tsit5 (Tsitouras 2011). While this is due to Tsit5 being more efficient for non-stiff problems in comparison to the IDA solver of OpenModelica 1.18.0, these results indicate that MTK can compete with OMC in terms of simulation performance. Still, while we were able to generate MTK code for up to 25,600 equations and variables, we experienced an exponential increase in compilation time when $N > 3200$ the reason for this behavior seems to be performance issues during the symbolic transformations of MTK v5.16.0.

6 Surrogate-based Optimization

The use of surrogates/metamodeling to accelerate computationally heavy models is not new. The idea is to replace computationally expensive components of a model with a surrogate model/metamodel to reduce computation cost and consequently reducing the feedback loop for modelers (Wang and Shan 2006).

In the context of Julia, Yingbo et al. have previously shown how to accelerate models by employing surrogates using MTK, where they employed surrogates to accelerate a Heating, ventilation, and air conditioning (HVAC) model claiming a 590X speedup compared to Dymola (Ma et al. 2021). This suggests that the ability to generate surrogates in a Modelica Compiler is a useful feature for users.

In this paper, we introduced a new backend in our Julia-based Modelica Compiler, the MTK backend. Consequently, it is able to use the surrogate facilities of MTK. In Listing 6 we illustrate how a surrogate can be generated from one of the models used in section 5 with *Surrogates.jl*[13]. In this example, we translate a Modelica model into an equivalent MTK model. We then create a radial surrogate of this model based on 30 samples, and the resulting simulation can be seen in Figure 5.

It is possible to use other software to generate surrogates based on Modelica models within a Julia environment. However, a novelty of the work presented here is the ability to generate surrogates for internal equations during compilation time. One application is employing surrogates for computational expensive external functions or (non)linear loops. Another feature could be to introduce a Modelica annotation `Surrogatize` to indicate to the compiler to automatically replace that component with a suitable surrogate.

6.1 Employing surrogates in the context of solving nonlinear systems of equations

In this subsection, we demonstrate how such a nonlinear algebraic loop can be replaced with a surrogate.

Listing 4. The Cascaded first order system from the scalable testsuite (Casella 2015).

```
model CascadedFirstOrder
  "N cascaded first order systems,
     approximating a pure delay"
  parameter Integer N = 10 "Order of the
     system";
  parameter Real T = 1 "System delay";
  final parameter Real tau = T/N "
     Individual time constant";
  Real x[N] (each start = 0, each fixed =
     true);
equation
  tau*der(x[1]) = 1 - x[1];
  for i in 2:N loop
    tau*der(x[i]) = x[i-1] - x[i];
  end for;
end CascadedFirstOrder;
```

[13] On Github:On GitHub: SciML/Surrogates.jl

Listing 5. Automatically generated MTK version of the Modelica code in Listing 4.

```
using ModelingToolkit
using DiffEqBase
using DifferentialEquations
function Casc10Model(tspan = (0.0, 1.0))
  pars = @parameters((T, tau, N, t))
  vars = @variables((x7(t),
                     x1(t),
                     x10(t),
                     x3(t),
                     x2(t),
                     x8(t),
                     x9(t),
                     x4(t),
                     x5(t),
                     x6(t)))
  der = Differential(t)
  eqs = [der(x7) ~ (tau^-1)*(x6 - x7), der(
      x1) ~ (tau^-1)*(1.0 - x1),
         der(x10) ~ (tau^-1)*(x9 - x10),
             der(x3) ~ (tau^-1)*(x2 - x3),
         der(x2) ~ (tau^-1)*(x1 - x2), der(
             x8) ~ (tau^-1)*(x7 - x8),
         der(x9) ~ (tau^-1)*(x8 - x9), der(
             x4) ~ (tau^-1)*(x3 - x4),
         der(x5) ~ (tau^-1)*(x4 - x5), der(
             x6) ~ (tau^-1)*(x5 - x6)]
  nonLinearSystem = ODESystem(eqs, t, vars,
                              pars,
                              name = :($(
                                  Symbol("
                                  Casc10"))
                                  ))
  pars = Dict(T => float(1.0),
              tau => float(T / float(N)),
              N => float(10), t => tspan
                  [1])
  initialValues = [x7 => 0.0, x1 => 0.0,
                   x10 => 0.0, x3 => 0.0,
                   x2 => 0.0, x8 => 0.0,
                   x9 => 0.0, x4 => 0.0,
                   x5 => 0.0, x6 => 0.0]
  firstOrderSystem = ode_order_lowering(
      nonLinearSystem)
  reducedSystem = dae_index_lowering(
      firstOrderSystem)
  problem = ODEProblem(reducedSystem,
      initialValues, tspan, pars)
  return problem
end
Casc10Model_problem = Casc10Model()
function Casc10Simulate(tspan = (0.0, 1.0))
  solve(Casc10Model_problem, tspan = tspan)
end
function Casc10Simulate(tspan = (0.0, 1.0);
     solver = Tsit5())
  solve(Casc10Model_problem, tspan = tspan,
       solver)
end
```

Listing 6. An example on how to automatically create a surrogate for Casc400 via a Julia script.

```
modelName = "Casc400"
n_sample = 30
surrogateFunction = (x, y, startTime,
    stopTime) -> Surrogates.RadialBasis(x,y
    ,startTime,stopTime)
#= Use backend target =#
ast = OMFrontend.parseFile("./Models/$(
    modelName).mo")
scodeProgram = OMFrontend.translateToSCode(
    ast)
(dae, cache) = OMFrontend.
    instantiateSCodeToDAE(modelName,
    scodeProgram)
OMBackend.translate(dae; BackendMode =
    OMBackend.MODELING_TOOLKIT_MODE)
#= Run Modelica model =#
omResult = OMBackend.simulateModel(
    modelName, tspan = (0.0, 1.0))
solution = getSolution(omResult)
#= Create surrogates for all states =#
x = sample(n_sample, startTime, stopTime,
    SobolSample())
y = omResult.(x)
surrogates = populateSurrogateArray()
#= Evaluate surrogates =#
surrogateResult = Array{Float64}(undef,
    modelN, length(omResult.u))
for i = 1:length(stateVars)
  surrogateResult[i,:] = surrogates[i].(
      omResult.t)
end
```

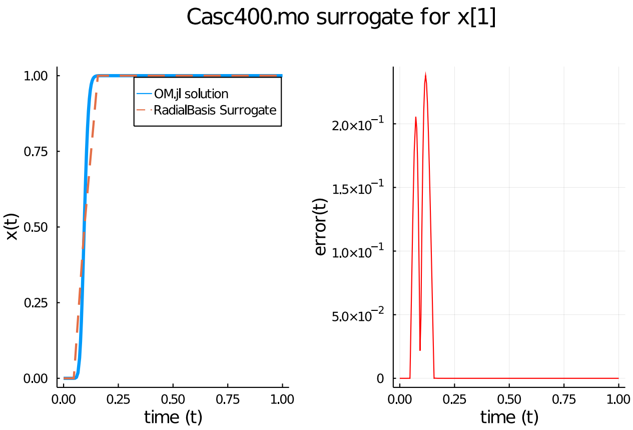


Figure 5. The graph to the left depicts the result of evaluating a Radial surrogate generated for the CascadedFirstOrder model with $N = 400$. The true function is represented in blue, the result of evaluating the surrogate is depicted in red. The right graph depicts the error as a function of time.

Listing 7. Modelica model where large parts of the simulation time are needed to solve a nonlinear loop.

```
model nonLinearScalable
  parameter Real a = 0.5;
  parameter Integer N = 10;
  Real x[N](each start=2.5);
  Real y(start=0,fixed=true);
equation
  for i in 1:N loop
    N+1 = exp(time*i*a+x[i]) + sum(x);
  end for;
  der(y) = sum(x)*time;
end nonLinearScalable;
```

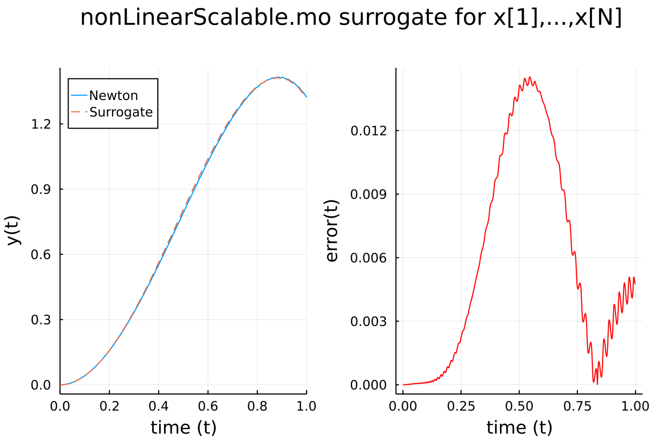


Figure 6. Simulation of nonLinearScalable using OMBackend and our surrogate function. The error is shown in red.

For the Modelica model nonLinearScalable, displayed in Listing 7, there is a dense nonlinear loop in variables $x_1,\dots,x_N$, that needs to be solved in each integration step to calculate the state variable y. The system is scalable in N.

During compilation, the loop is detected, and we collect all equations and variables involved in the loop to generate a function to solve the algebraic loop. For this example, we manually generated training data by solving the loop at time points in $[0,1]$ with a Newton method. Then a small neural network was trained on these solutions to replace the Newton solver normally used to solve the nonlinear algebraic system in the function *nonLinearScalableModel_surrogateODE*.

The speedup of the simulation time with the surrogate compared to simulating with a nonlinear solver that uses the Newton method was approximately 163.2 while the memory consumption was reduced from around 10.6 MiB to 0.4 MiB.

The solution of state variable y of `nonLinearScalable` where the nonlinear equation system is solved with Newton's method and with a surrogate can be studied in Figure 6. Because a simple structure for the neural network was chosen, some accuracy was lost while speeding up the simulation significantly.

To conclude, this integration enables the possibility of

Listing 8. The generated Julia code corresponding to listing 7. Some equations in the nonlinear system are left but are available upon request.

```
function nonLinearScalableAlgebraicLoop()
  parameters = ModelingToolkit.@parameters
      ((a, N, t))
  vars = ModelingToolkit.@variables((y(t),
      x1, x2, x3, x4, x5, x6, x7, x8, x9, x
      10))
  eqs = [
        0 ~ 0.0 - (((t * a + x1) + (x10 +
            (x9 + (x8 + (x7 + (x6 + (x5 +
            (x4 + (x3 + (x2 + x1))))))))))
            - float(N + 1)),
        ....]
  nonLinearSystem = ModelingToolkit.
      NonlinearSystem(eqs, vars, parameters
      , name = :($(Symbol("
      nonLinearScalable"))))
  return nonLinearSystem
  end
function makeNLProblem()
  loop = nonLinearScalableAlgebraicLoop()
  nlsys_function = (generate_function(loop,
       expression = Val{false}))[2]
end

nonLinearScalableNonLinearFunction =
    makeNLProblem()
function nonLinearScalableModel_ODE(dx, x,
    aux, t)
  p = aux[1]
  u = aux[2]
  func!(res, u) =
      nonLinearScalableNonLinearFunction(
      res, u, vcat(p, [t]))
  sol = nlsolve(func!, u, ftol = 1.0e-12;
      method = :newton)
  aux[2] = sol.zero
  dx[1] = (u[8] + u[9] + u[7] + u[6] + u[5]
       + u[4] + u[3] + u[2] + u[11] + u
      [10]) * t
end

function
    nonLinearScalableModel_surrogateODE(dx,
     x, aux, t)
  u = aux[2]
  aux[2] = m([t])
  dx[1] = (u[8] + u[9] + u[7] + u[6] + u[5]
       + u[4] + u[3] + u[2] + u[11] + u
      [10]) * t
end
```

auto-tuning Modelica models or parts of such models to accelerate simulation time in a fashion similar to that of Ma et al. (2021). This trade-off between accuracy and simulation time may be suitable for industrial applications, e.g. automatically reducing the high detailed development model to a real-time capable surrogate running on an integrated chip.

7 Conclusion and Future Work

The results of this paper indicate the feasibility of a Modelica Compiler in Julia. Concerning the experiments in section 5, we acknowledge that there is a wide range of different solvers available in MTK and in OpenModelica. Thus, the goal of these experiments is not to assess the performance of handwritten MTK-models but rather to characterize the performance of automatically generated MTK-models in the context of the compiler presented here.

Furthermore, having access to the ModelingToolkit ecosystem enables several extensions. One such extension is the possibility to generate surrogates during compilation time automatically. Challenges for such a scheme include not only the selection of surrogatisation techniques but also how to decide what part of a model to replace and what kind of surrogate to employ.

Another direction for future work would be to examine further the application of surrogatisation techniques in the context of algebraic loops or whole sub-models. That is replacing algebraic loops in large industrial grade DAE systems with suitable surrogates during compilation time. While we presented some initial examples as a part of this paper, further work is required to establish the efficiency of such techniques. To conclude, in this paper, we present OpenModelica.jl, a non-monolithic Modelica Compiler written in Julia using the high-performance frontend from the OMC that can connect the Modelica ecosystem with the ecosystem of Julia and ModelingToolkit.

Acknowledgements

This work has been supported by SSF in the LARGEDYN project. This work has also been supported by Vinnova in the ITEA EMPHYSIS project and the EMISYS project. Support from the Swedish Government has also been received through the ELLIIT project. OpenModelica development is supported by the Open Source Modelica Consortium. Many students, researchers, and engineers have contributed to the OpenModelica environment. There is not enough room here to mention all these people, but we gratefully acknowledge their contributions. We would also like to thank the reviewers for their comments to improve this paper.

References

Bezanson, Jeff et al. (2017). "Julia: A fresh approach to numerical computing". In: *SIAM review* 59.1, pp. 65–98.

Braun, Willi, Francesco Casella, Bernhard Bachmann, et al. (2017). "Solving large-scale Modelica models: new approaches and experimental results using OpenModelica". In: *12 International Modelica Conference*. Linkoping University Electronic Press, pp. 557–563.

Casella, Francesco (2015). "Simulation of large-scale models in modelica: State of the art and future perspectives". In: *11th International Modelica Conference*, pp. 459–468.

Chen, Jiahao and Jarrett Revels (2016-08). "Robust benchmarking in noisy environments". In: *arXiv e-prints*, arXiv:1608.04295. arXiv: 1608.04295 [cs.PF].

Elmqvist, Hilding and Martin Otter (2017). "Innovations for future Modelica". In: *Proceedings of 12th International Modelica Conference*. Linköping University Electronic Press.

Fritzson, Peter, Adrian Pop, Karim Abdelhak, et al. (2020). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control* 41.4, pp. 241–295.

Fritzson, Peter, Adrian Pop, and Peter Aronsson (2005). "Towards Comprehensive Meta-Modeling and Meta-Programming Capabilities in Modelica". In: *Proceedings of the 4th International Modelica Conference, Hamburg, Germany*. Citeseer.

Hindmarsh, Alan C et al. (2005). "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers". In: *ACM Transactions on Mathematical Software (TOMS)* 31.3, pp. 363–396.

Ma, Yingbo et al. (2021). *ModelingToolkit: A Composable Graph Transformation System For Equation-Based Modeling*. arXiv: 2103.05244 [cs.MS].

Parr, Terence J. and Russell W. Quong (1995). "ANTLR: A predicated-LL (k) parser generator". In: *Software: Practice and Experience* 25.7, pp. 789–810.

Pop, Adrian et al. (2019). "A New OpenModelica Compiler High Performance Frontend". In: *13th International Modelica Conference*. Vol. 157, pp. 689–698.

Rackauckas, Christopher and Qing Nie (2017). "Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia". In: *Journal of Open Research Software* 5.1.

Rackauckas, Christopher and Qing Nie (2019). "Confederated modular differential equation APIs for accelerated algorithm development and benchmarking". In: *Advances in Engineering Software* 132, pp. 1–6.

Sarı, Zekeriya and Serkan Günel (2019). "Causal. jl: A Modeling and Simulation Framework for Causal Models". In: *Proceedings of JuliaCon* 1, p. 1.

Tinnerholm, John et al. (2020). "Towards an Open-Source Modelica Compiler in Julia". In: *Proceedings of Asian Modelica Conference 2020, Tokyo, Japan, October 08-09, 2020*, pp. 143–151. DOI: 10.3384/ecp20174143.

Tsitouras, Ch (2011). "Runge–Kutta pairs of order 5 (4) satisfying only the first column simplifying assumption". In: *Computers & Mathematics with Applications* 62.2, pp. 770–775.

Wang, G Gary and S Shan (2006). "Review of metamodeling techniques in support of engineering design optimization". In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. Vol. 4255, pp. 415–426.

Investigating Steady State Initialization for Modelica models

Hans Olsson[1] Erik Henningsson[2]

[1] Dassault Systemes AB, Sweden, hans.olsson@3ds.com

[2]Dassault Systemes AB, Sweden, erik.henningsson@3ds.com

Abstract

This paper investigates steady-state initialization both symbolically and numerically, and in particular demonstrates new ways of adapting the symbolic methods for finding steady-state solutions for Modelica models extending ideas that were previously manually implemented in libraries. The methods are compared on realistic Modelica models in Dymola.

Keywords: initialization, fluid models, differential algebraic equation, static simulation, steady state

1 Introduction

Steady state behavior for models is an important study – for several reasons. The steady state solution can be the goal of the particular study, when studying how parameters influence its characteristics, – and also a starting point for normal simulation studies, since starting from a steady-solution avoids uninteresting transients.

The goal of this paper is to investigate strategies for finding steady states of Modelica models. We survey different approaches and discuss their benefits and challenges. Additionally, we present strategies for automatic treatment of structural and numerical singularities arising due to the steady-state formulation. Such problems are for example encountered in fluid models. To the best of our knowledge, such automatic handling specifically for steady-state initialization has not previously been discussed in the literature even if several of the methods have been discussed.

In addition to true steady-state solutions where all derivatives are zero and the solution is unchanging, we as an outlook consider have quasi steady-state where some states are changing, but the solution is fundamentally time-invariant – e.g., a vehicle running at a constant velocity. This creates unique challenges that will be discussed later.

The stability of the steady-state solution could be analyzed by a linearization at that point, but we will not consider it in detail.

2 Variants of initialization

Normally Modelica models are initialized using the model equations, initial equations, fixed start-values as described in (Mattsson 2002), and supplemented by selecting non-fixed start-values as described in section 8.6.2 “Recommended selection of start-values” of Modelica 3.5 (Olsson (editor), 2021).

The problems are often numerically challenging and homotopy methods can be useful for handling that; see (Sielemann 2011), (Sielemann 2012), and (Casella 2011).

Note that the initialization is applied after the index-reduction, and we will thus primarily consider the ODE or index-1 DAE-formulation of the model that due to index-reduction may require additional conditions. Note that historically the index reduction algorithm, (Pantelides 1988) was proposed specifically to find those initial conditions.

Note that the difference between the ODE and index-1 formulation is important here as there might be initial conditions and/or start-values involving the algebraic variables. Additionally, index reduction sometimes lead to dynamic state selection, (Mattsson 2000) and the combination with steady state initialization poses specific problems that will be considered later.

Various libraries have different mechanisms for conditionally disabling initial equations and start-values.

2.1 Symbolic steady state initialization

Steady state initialization changes this to instead of selecting start-values we set derivatives to zero. This might be seen as simply changing from computing a solution $x(t)$ from (this is most easily seen in the ODE case):

$$\begin{gathered}\dot{x} = f(x(t), t) \\ x(0) = x_0\end{gathered} \tag{1}$$

to computing it from:

$$\begin{gathered}\dot{x}(t) = f(x(t), t) \\ 0 = \dot{x}(0)\end{gathered} \tag{2}$$

However, even if this works in some cases there are number of issues in most cases:

- The Jacobian $\partial f/\partial x$ might be singular, either structurally or just at the initial point; indicating that there are multiple acceptable steady-state solutions; this is discussed in (Casella 2012). It can be difficult to give good numeric diagnostics for this, and we will return to this in Section 3.

- Some initial conditions might be specified, either to handle the singularity above or in order to get a specific quasi-steady-state solution.

These problems can be overcome using specific remedies as will be explained later.

2.2 Dynamic steady state finding

Another alternative is to numerically integrate forward from the normal initial conditions until a dynamic steady-state is reached (up to a certain precision). This is often simpler, and handles the issues above. Furthermore, by introducing pseudo dynamics, purely static models can be transformed to dynamic models. The dynamics are crafted so that the solution tends to a steady state equal to the static solution of the original model. This technique is often employed to break up algebraic loops for more robust solving of static problems. For both of these application it is important to automatically detect that the steady state has been reached so that the simulation can be terminated in a timely fashion.

However, there are other potential downsides when using the dynamic steady state initialization:

- The dynamic simulation usually takes longer than the static initialization.
- Double integrators – especially in combination with "quasi-steady-state", will tend to infinity.
- The model may not have a steady-state solution; but instead tend to a periodic solution, quasi-periodic, or even have a strange attractor.
- The solution may have state events during the solution. There might also be time events if integrating forward in time.

Methods to detect periodic steady states have previously been considered in the Modelica context. (Kuhn 2017) investigates techniques to automatically identify periodic steady states in Modelica models of electrical AC systems. An additional example is the Electrified Powertrains Library (EPTL), which feature components that terminate a simulation when a periodic steady state is reached.

It might be possible to avoid time events and periodic solutions by integrating from minus infinity towards zero, and use implicit Euler with large step-sizes to artificially dampen oscillations.

3 Symbolic steady state initialization

We will now consider the specific issues related to symbolic steady state initialization, and our general approach for solving it. Specifically we will describe how the changes to the basic approach handles various cases.

3.1 Outline of symbolic approach

The goal of the symbolic approach is to set up the modified steady-state problem to ensure that it is structurally non-singular, and numerically non-singular at the desired steady state solution.

We start from $\dot{x} = f(x(t), t)$ but instead of setting the entire $\dot{x}(0) = 0$ we use the well-known maximum-bipartite-matching; e.g., (Cormen 1990) for finding which elements of $\dot{x}$ that should be set to zero.

The matching is similar to the matching for transient simulation where we require a perfect (one-to-one) matching of derivatives to these equations. The matched variables and equations are also sorted into strongly connected components and each of them solved separately; but we will not discuss that in detail.

However, for initialization we instead attempt to match variables to $\dot{x} = f(x(t), t)$, prioritized to first match states and then derivatives of states so that all equations are matched to some variable – but not all variables is matched to an equation.

The states and derivatives of states that were unmatched in this initialization problem are then set to default values (normally zero for derivatives) and start-value for states. This matching is based on the variable incidences for each equation, but modified as will be explained later.

This procedure is then amended for an index-1 DAE by adding algebraic equations (and initial equations) and first matching all auxiliary variables.

An important aspect is that when matching states to the equations we prioritize them to get the result of section 8.6.2 "Recommended selection of start-values" of Modelica 3.5 (Olsson (editor) 2021).

Note that this is only used for solving the initialization problem and we then use the equations in the usual way for dynamic simulations.

This basic approach was implemented in Dymola in 2004 based on the similarity with the normal state-initialization. There are multiple variants of this; note that we symbolically manipulate the equations but do not aim to symbolically solve them in contrast to (Ochel 2014).

3.2 Avoiding structural singularities

For fluid models we have seen two specific causes of singularities. The first issue for fluid models, already handled in models in (Casella 2012), can be simplified to two tanks connected in a cycle where the outflow, f, from each tank depend on its mass, x, and some parameter A:

$$\begin{gathered} f_j = g(x_j,\, A_j) \\ \dot{x}_1 = f_2 - f_1 \\ \dot{x}_2 = f_1 - f_2 \end{gathered} \tag{3}$$

This can be integrated forward in time, but if we attempt to compute the steady-state solution we get a redundant equation:

$$\begin{gathered} 0 = f_2 - f_1 \\ 0 = f_1 - f_2 \end{gathered} \tag{4}$$

The proposed solution for such cases is to consider all trivial equations when derivatives are set to zero, and see if they form cycles (as in this case). Trivial equations are equations that can be written on the form a=+/-b. This automates the procedure from (Casella 2012).

As previously indicated the symbolic selection of initial conditions is based on matching variables to equations based on their incidence. We thus modify the equation graph causing the cycle (in this case the two derivative-equations) by adding an incidence to an extra algebraic variable in them and introducing a new equation with incidence to the derivative-variables of these equations (the variable is called h below; the equation is left empty) which gives the graph:

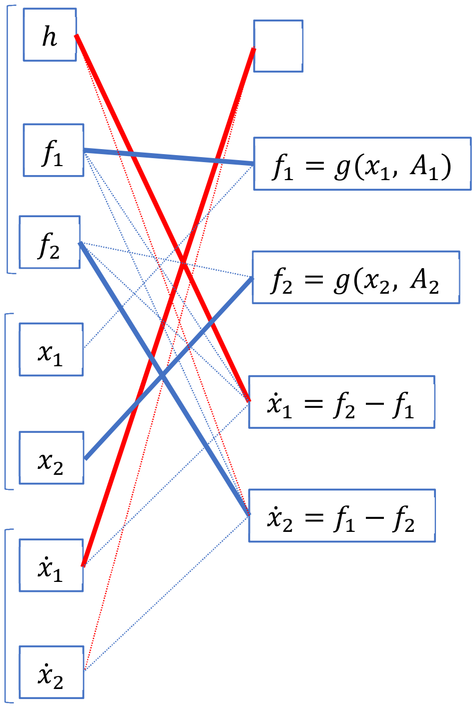


Figure 1 Matching for two tanks

The variables are in three groups, since we first match h, f_1, f_2; then attempt to match x_1, x_2 and finally attempt to match $\dot{x}_1, \dot{x}_2$. The matchings are solid lines and other incidences are dotted thinner lines.

The unmatched derivative $\dot{x}_2$ is set to zero, and the unmatched state x_1 becomes an arbitrary start-value (there is a priority depending on whether start-values are set and at what level, such that the start-value that is "highest up" is prioritized). The other state is initialized based on $f_1 = f_2$ giving $g(x_1, A_1) = g(x_2, A_2)$.

We then modify the new matching to remove the extra equation and extra variable h, but keep the previously matched variables. (There might be other ways of getting to this desired matching without temporarily adding an extra variable and equation and then removing them). This matches $\dot{x}_1$ with $\dot{x}_1 = f_2 - f_1$; and the equations are then as usual sorted into strongly connected components where each component in this case is scalar; and only the one involving x_2 require the solution of a non-linear equation.

From a modeling perspective another possibility would be to have an initial equation for the total mass in the system, replacing the arbitrary start-value for one of the masses. The chosen approach naturally handles that, and the initial equation can either be conditional on steady-state initialization or always applied.

3.3 Avoiding singularity at the solution

The second issue for fluid models can occur for any differentiated media such as the function for density as a function of the states:

$$\rho = d(T, p) \tag{5}$$

Which is differentiated to give:

$$\dot{\rho} = d_T(T, p)\dot{T} + d_p(T, p)\dot{p} \tag{6}$$

If we just solve the equations without symbolic processing as suggested in (Casella 2012) this is unproblematic.

However, if we want to perform a structural analysis to see which derivatives we can set to zero we get a problem. Specifically we might attempt to match T or p to this equation during steady initialization and attempt to compute them from it – because they influence $d_T(T, p)$ and $d_p(T, p)$.

Numerically this will not be well-behaved when we approach the steady-state solution since the influence of T and p on $d_T(T, p)\dot{T} + d_p(T, p)\dot{p}$ gradually disappear as the derivatives approach zero.

We handle that by modifying the incidence for all initialization equations by removing checking what happens if the state-derivatives are set to zero. If the equation no longer contains a non-derivative variable we remove the incidence from the equation to that variable when finding the steady-state solution.

3.4 Higher order derivatives

The original description separated variables into states and derivatives of states. For higher order derivatives that naturally occur in mechanical systems that is not always straightforward -- consider a simple rotational model:

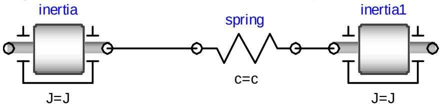


Figure 2 Two inertias

The rotational speed, w, is both a state and a derivative; and it may even have a start-value.

However, the state-variable w can be matched to the equation der(phi)=w; i.e. it will be treated as if it only were a derivative-variable. Thus such higher order derivatives cause no direct problem, and since the state is matched to equations its start-value is ignored (assuming there is no fixed=true). We might in the future prioritize start-values for such combinations of state/derivative instead of setting derivatives to zero, but other related issues are more important.

3.5 Dynamic State Selection

Consider the pendulum in Cartesian coordinates.

```
model Pendulum
  Real vx,vy;
  Real x(start=0.5),y(start=0.7);
  Real f;
equation
  x^2+y^2=1;
  der(x)=vx;
  der(y)=vy;
  der(vy)=-9.81+f*y;
  der(vx)=f*x;
end Pendulum;
```

Note that the start-values are inconsistent guess-values.

In Modelica tools the pendulum is usually solved using the dynamic dummy derivative method, since there is no static selection of states that generate a good solution.

Note that there are two dynamic dummy derivative systems, each selecting one state among two possibility – one for positions and one for velocities.

However, that is an implementation detail and from a user perspective we prefer to hide those dummy variables and instead give start-values for the normal variables. Using high order derivatives all differentiated equations can be written as:

$$x^2 + y^2 = 1 \tag{7a}$$

$$2\dot{x}x + 2\dot{y}y = 0 \tag{7b}$$

$$2\ddot{x}x + 2\dot{x}^2 + 2\ddot{y}y + 2\dot{y}^2 = 0 \tag{7c}$$

$$\ddot{y} = -9.81 + fy \tag{7d}$$

$$\ddot{x} = fx \tag{7e}$$

Simply ignore the dummy derivatives?

Hiding the dummies causes two problems: the first is that it looks as if we need two steady-state conditions and have four free derivatives (two velocities and two accelerations) that we could set to zero for steady-state initialization, but in reality there are only two free derivatives in total (one velocity and one acceleration).

If we ignore that and set the two velocities to zero that directly collapses one existing equation:

$$2\dot{x}x + 2\dot{y}y = 0 \xrightarrow{yields} 0 = 0 \tag{8}$$

Similarly setting the two accelerations to zero simplifies another existing equation:

$$2\ddot{x}x + 2\dot{x}^2 + 2\ddot{y}y + 2\dot{y}^2 = 0 \xrightarrow{yields} 2\dot{x}^2 + 2\dot{y}^2 = 0 \tag{9}$$

Which have the following Jacobian with respect to the velocities:

$$\begin{bmatrix} 2x & 2y \\ 4\dot{x} & 4\dot{y} \end{bmatrix} \tag{10}$$

When the velocities goes to zero the second row tends to zero.

Solving that problem leads to the second problem, that some choices of velocity and acceleration-variables for steady state lead to singular systems, which is exactly the reason we used dynamic dummy-derivatives in the first place. It is not certain that a dummy-derivative system is singular at exactly the steady state solution, but it seems likely and will occur in this case if we set

$$\ddot{y} = \dot{y} = 0 \tag{11}$$

which gives the following Jacobian with respect to $x, y, \dot{x}, \ddot{x}, f$ (where the second and third rows are zero at the solution):

$$\begin{bmatrix} 2x & 2y & 0 & 0 & 0 \\ 2\dot{x} & 0 & 2x & 0 & 0 \\ 2\ddot{x} & 0 & 4\dot{x} & 2x & 0 \\ 0 & -f & 0 & 0 & -y \\ -f & 0 & 0 & 1 & -x \end{bmatrix} \tag{12}$$

Directly using the dummy derivatives

An alternative would then to be to attempt to set the actual derivatives of the dynamic dummy derivative method to zero. But that seems underspecified, since it involves projecting the original derivatives using an unspecified projection matrix.

This approach might work using an iterative scheme where the projection matrix is recomputed until convergence, but it seems needlessly complicated and it is not obvious that it will generate a unique solution.

Proposed solution

The proposed remedy here involves modifying the initial equations in a consistent way.

First we set *all* derivatives in each dummy derivative system to zero as if we ignore the dummy derivatives, and then we balance that by removing the corresponding differentiated constraint that becomes identically zero for that choice (after verifying that it is in fact the case).

Additionally we propagate these zeros to other dummy derivative systems, since normally the acceleration constraints also involve velocities.

For the pendulum this means that we want to solve:

$$x^2 + y^2 = 1 \tag{13a}$$

$$\dot{x} = 0 \tag{13b}$$

$$\dot{y} = 0 \tag{13c}$$

$$\ddot{y} = -9.81 + fy \tag{13d}$$

$$\ddot{x} = fx \tag{13e}$$

$$\ddot{x} = 0 \tag{13f}$$

$$\ddot{y} = 0 \tag{13g}$$

and verify that the differentiated constraints are zero:

$$2\dot{x}x + 2\dot{y}y = 0 \tag{14a}$$

$$2\ddot{x}x + 2\dot{x}^2 + 2\ddot{y}y + 2\dot{y}^2 = 0 \tag{14b}$$

4 Future work: Quasi Steady State

Quasi steady state, where some derivatives have non-zero values is important in practice. The goal would be to directly study the model in operation without unnecessary transients. A realistic example would be a model of a car running at 60 km/h.

The goal of this section is two-fold; both to indicate how a tool in the future could automatically find these solutions, and also to demonstrate the problem to avoid the unintended solutions shown here.

4.1 Why quasi steady state is complicated

Let us start by a simple example of two connected inertias.

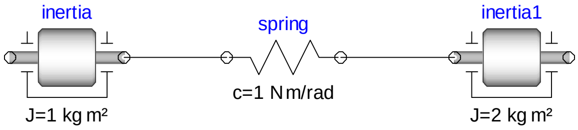


Figure 3 Two Inertias, Quasi Steady State

Assuming we want quasi steady-state and set inertia.w(start=5, fixed=true); something odd happens in this example. Just setting all other derivatives to zero makes the other inertia starts stationary and then we get a periodic solution for the angular velocities.

If we use a damper (or spring-damper) the states will as default be in the damper and thus as default the damper will have derivative zero, but if we set options to avoid states in the damper there is a risk of a similar non quasi-steady-state solution.

Note that it is not necessarily that we give a rotational velocity – a more realistic case is that we attach an engine (can even be a simple constant torque) to one inertia and some losses such as bearing-friction to the other. The result is the same, we can set one acceleration to zero, but not the other; and the solution will then start with a similar transient until reaching a quasi steady state.

Obviously this is not the desired quasi-steady solution, and the idea with investigating a small example is to find an approach that can be applied to a large system, like a car.

One approach is that instead of selecting initial conditions among the existing derivatives we add the initial equations $\dot{w} = 0$ and $\ddot{w} = 0$, and that is similar to treating w=5 as a normal non-initial equation during index-reduction. However, if we had another spring (or spring-damper) followed by an inertia we would need to set $\dot{w} = 0$ and $\ddot{w} = 0$ for this new inertia, etc.

Partially this is just prioritizing setting high order derivatives to zero, i.e. $\dot{w} = 0$ instead of $w = 0$, but $\dddot{w}$ does not even exist in the model, and even constructing $\dddot{w}$ in the first inertia does not create the next one. Thus further investigations are needed. We can also consider a clutch instead of a spring-damper, and a solution can have the two inertias rotating with different (steady-state) speeds.

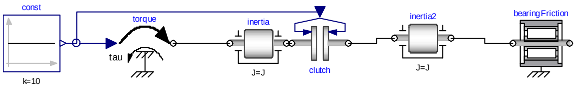


Figure 4 Advanced Quasi Steady State

An additional complication occurs if there are multiple disconnected mechanical systems – the initialization procedure outlined here will allow them to move independently of each other.

This also indicates that the original name we used for the approach "DefaultSteadyState" is misleading, since using steady state as "default" for a few variables does not guarantee a meaningful result.

4.2 Simple quasi steady state theory

Assume we have a system:

$$\dot{x} = f(x) \qquad (15a)$$

and we want to generalize the notion steady state to cases where the derivative is non-zero but the system does not change. The simplest possibility is that the derivative is constant which gives that the following should be valid for all points in time:

$$\dot{x} = f(x + \dot{x}t) \qquad (15b)$$

This implies that the derivative is in the non-trivial null-space

$$\ker\left(\frac{\partial f}{\partial x}\right) \qquad (16)$$

and such null-spaces exist if the model is translationally invariant, such that y=T(x,p) behaves the same as x, with the restriction that the transformation has determinant 1 (and is differentiable in p). In general quasi steady state meaning that all points on the trajectory are "equivalent" seems like a good definition for quasi steady state, and smooth transformations allow that, and for a general fixed transformation we have

$$\dot{x} = \left(\frac{\partial T}{\partial x}\right)^{-1} f\big(T(x,p)\big) \qquad (17)$$

For translational invariance $\left(\frac{\partial T}{\partial x}\right) = I$ and we can select a trajectory as a varying transformation of one point (since it always gives the same derivative). The restriction is then that the derivatives are given as $\frac{\partial T}{\partial p}$.

The benefit of this formulation can be seen for second order 1D-mechanical systems where we directly see that velocities should be in this null-space. Compare this with setting $\ddot{w} = 0$ (i.e., setting the third order derivative of the angle to zero) that require differentiating the model equations an additional time. Both formulations ensure quasi steady state behavior.

The null-space can be fairly simple, e.g. all absolute positions (with equal weight), and without relative positions. If we have a rotational motion with gears it is less trivial, but still straightforward.

4.3 Multi-dimensional fixed speed rotations

However, if we want to consider a car turning with a fixed steering angle (i.e. with a constant yaw) it becomes complicated. In general we propose the equation:

$$\left(\frac{\partial T(x_0,p)}{\partial p}\right)\dot{p} = f\big(T(x_0,p)\big) \qquad (18)$$

The desired solution for rotation can be seen a pure rotation – but around an unknown point in space, and for the positions the second derivatives are thus non-zero (and proportional to the distance from the rotation center for absolute coordinates). If we use $p = \{r_x, r_y, \theta\}$ as invariant position and angle of the main object and then relative coordinates, x, we have the invariants:

$$\ddot{r} = f\big(\dot{r},\theta,\dot{\theta},x,\dot{x}\big) = e^{R\theta} f\big(e^{-R\theta}\dot{r},0,\dot{\theta},x,\dot{x}\big) \qquad (19a)$$

$$\ddot{\theta} = g\big(\dot{\theta},x,\dot{x}\big) \qquad (19b)$$

$$\ddot{x} = h\big(\dot{\theta},x,\dot{x}\big) \qquad (19c)$$

with the solution $\ddot{x} = 0$, $\dot{x} = 0$, $\ddot{\theta} = 0$, $\ddot{r} = \dot{r}R\dot{\theta}$. For a rocket in space all positions and angles can be invariant, but for more earthly applications, such as a typical car (on a tarmac that is big, flat, and even), the positions in the plane are the invariant positions, and the yaw angle the invariant angle (the yaw velocity is $\dot{\theta}$).

This seems straightforward, but it is not clear how to perform this when only using the equations - without any additional knowledge. Additionally if the road surface depends on the position or is slanted, there are no solutions that rotate with a fixed speed.

However, this is the desired solution if we want to set the speed for the car (or have the engine running), and have the steering wheel off-center (or in general anything that makes the car non-symmetric) – i.e. it is the natural extension of the simple translational case for quasi steady state to 3-dimensions and also the special case of 2-dimensions (Höbinger and Otter, 2008).

Note that this is mostly restricted to mechanical systems, due to lack of invariants in other cases. Electrical systems are invariant with respect to the potentials – even more generally than mechanical systems (the potential can be any time-varying function), but grounding the circuit eliminates that – and grounding is used for normal simulation as well.

5 Evaluating the methods

There are three factors we want to consider for these methods:

- Does is find the desired steady-state solution?
- How easy is it to set up the problem?
- How quickly do we compute the solution?

5.1 Furuta pendulum

In order to dynamically find a steady state solution we currently have to add dampers to all joints to ensure that the steady state solution is asymptotically stable. And then set flag to find dynamic steady-state.

To generate the symbolic steady state solution the dampers in Figure 5 are not necessary (although possible), just setting the symbolic steady-state flags suffice. However, unless we set the guess-value for R2.phi close to zero this generates an unstable steady-state solution.

The symbolic handling automatically detects that R1.phi is a free variable, since the rotation axis is aligned with gravity. It is thus selected as a fixed start value in accordance with Section 3.2.

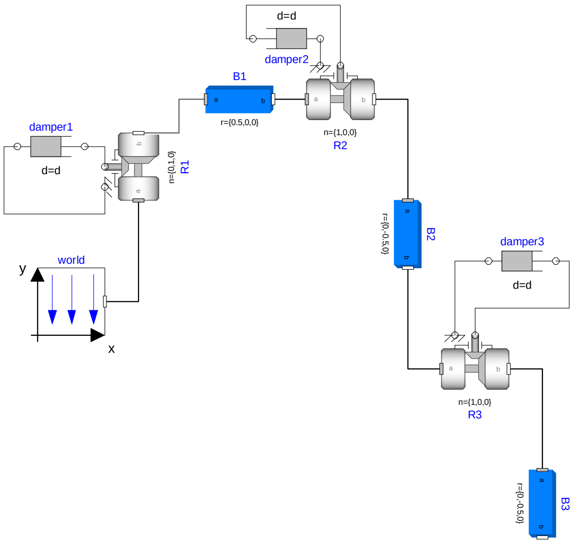


Figure 5 Damped Furuta Pendulum

5.2 Three tanks

This is a simple model in the Fluid package in the standard library.

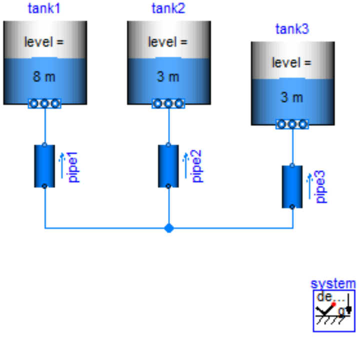


Figure 6 Three Tanks Steady State

Merely simulating the model gives the following plot (converging to a steady state):

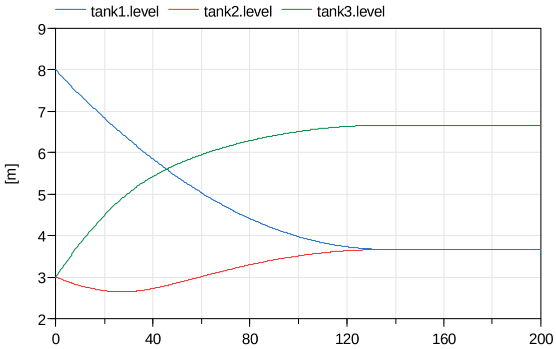


Figure 7 Solution for Three Tanks Steady State

There is a global setting for initialization in the model which has three relevant possibilities:

- FixedInitial – the default giving the plot
- Steady-state initial
- Initial guess-value

The initial guess-value in combination with steady-state flags finds a desired solution, and reports that one tank level and all tank temperatures are free initial values.

The built-in steady-state initial setting also finds a steady-state solution and reports the same free initial values, but additionally generate a diagnostic that there are four redundant consistent initial conditions are automatically removed:

```
  Removed the following equations which
are redundant and consistent:
der(tank3.U) = der(tank3.m)*tank3.medium.u
  + tank3.m*der(tank3.medium.u);
der(tank1.U) = der(tank1.m)*tank1.medium.u
  + tank1.m*der(tank1.medium.u);
der(tank2.U) = der(tank2.m)*tank2.medium.u
  + tank2.m*der(tank2.medium.u);
pipe1.port_a.m_flow+pipe2.port_a.m_flow
   +pipe3.port_a.m_flow = 0.0;
```

5.3 HeatLosses in MultiBody

This example model is interesting because it mixes different domains and has dynamic state selection.

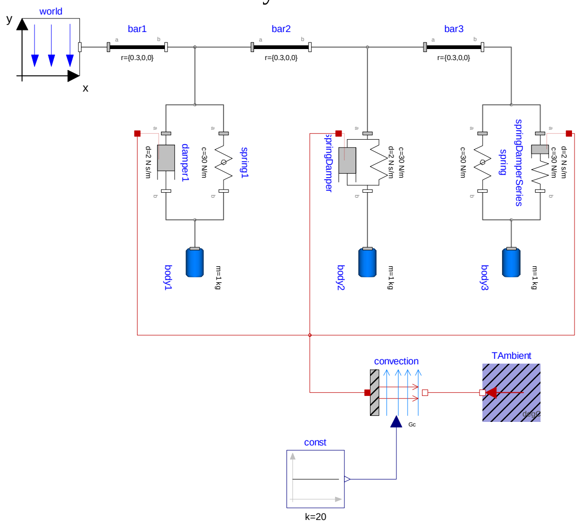


Figure 8 Multiple Springs with Heatlosses

The heat-losses are due to friction in the dampers, and should be zero in steady-state (confirmed by examining the solution at steady-state).

The symbolic steady-state settings directly finds the steady-state solution (the straight lines in the diagram) matching the dynamic solution.

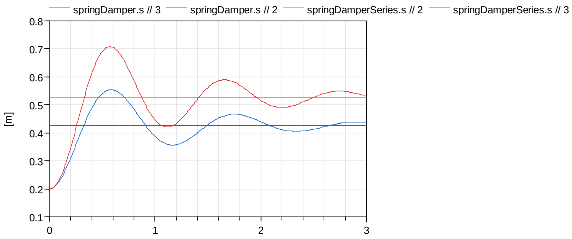


Figure 9 Solution for Multiple Springs with Heatlosses

The dynamic steady-state does not quickly find that the solution has converged, since there is a hidden slowly damped oscillation.

5.4 Quasi-steady state vehicle

Using constant torque for a simple translational vehicle with a non-rigidly attached trailer:

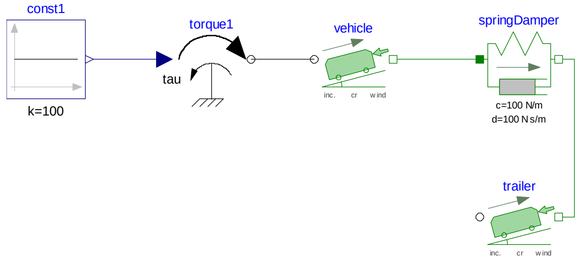


Figure 10 Quasi Steady-State for Simple Vehicle

Using dynamic steady state automatically finds the steady-state velocity (after 1 minute).

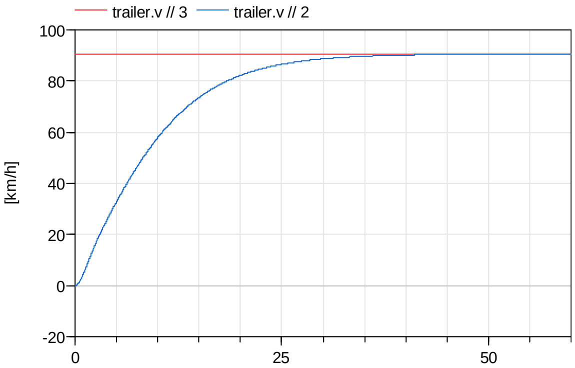


Figure 11 Solution to Quasi Steady-State for Simple Vehicle

Since the quasi-steady symbolic solution is not yet implemented we would have to manually add the corresponding initial equations based on Section 4.2 (the first two give that the derivatives are in the null-space and the final one is setting a second order derivative to zero):

```
  vehicle.v=trailer.v;
  vehicle.a=trailer.a;
  vehicle.a = 0;
```

directly giving the solid line above.

5.5 Implementation notes

The symbolic steady state initialization with the improvements listed in section 3 were implemented already in Dymola 2020, but Dymola 2022 and 3DExperience 2022x adds the possibility of ignoring some state initializations in the model which makes it easier to perform the tests. The flags used are:

```
Advanced.Translation.
 DefaultSteadyStateInitialization=false;
Advanced.Translation.
 DefaultSteadyStateInitializationLevel=1;
```

Finding the dynamic steady-state initialization was implemented in Dymola 2022, and is enabled by the flag:

```
Advanced.Simulation.
```

```
SteadyStateTermination=true;
```

By default the simulation is automatically terminated when all state derivatives have an absolute value less than 2 % of the scale of the corresponding state, taking into account the time scale of the simulation. This default tolerance is chosen in accordance with the common definition of *settling time* within control theory (Tay, Mareels and Moore 1998). The tolerance can be modified by

```
Advanced.Simulation.
  SteadyStateTerminationTolerance
```

for details see (Dassault Systèmes 2021).

5.6 Evaluation

Both methods quickly find a steady-state solution, and are fairly easy to set up.

The downside of finding dynamic steady-state is that one must ensure that the system is sufficiently damped to reach a steady state; whereas the downside of symbolic steady state is that it may find an unwanted steady-state solution.

6 Outlook for standardization

The new features are specific to Dymola, and not standard Modelica.

The Modelica Language has no features for switching between setting values for states and steady-state initialization. However, many Modelica models have such settings, locally and/or using an inner component to have a “global” setting.

The examples show that such settings, when they exist, work similarly as the tool-setting for symbolically solving the steady-state problem. This means that any standardization effort needs to ensure that the existing models can seamlessly switch to the new formulation which will be an additional effort.

One downside of having the steady-state setting in the model is that although index-reduction automatically handles algebraic couplings for the dynamic equations that does not automatically remove initialization equations. However, that is not a major issue as tools can detect such redundant initial equations and automatically remove them.

A specific issue with having steady-state settings locally in components is that this can easily set up quasi steady-state problems generating undesirable solutions as indicated above.

7 Conclusions

This paper demonstrates that Modelica allows powerful initialization techniques, both symbolic and numeric. The methods have been implemented in Dymola 2022 and 3DEXPERIENCE 2022x.

However, to be standardized in Modelica this must be integrated in models complementing the existing model settings and it must be possible to detect and preferably solve quasi steady state problems.

Acknowledgements

Some of this work has been part of the ModeliScale research project, and we also acknowledge the customers providing us with challenging models.

References

Dassault Systèmes. (2021) Dymola 2022: *Dymola, Dynamic Modeling Laboratory, User Manual 2A: Model Development Tools.* Dassault Systèmes AB, Lund, Sweden.

Casella, Francesco, and Michael Sielemann, and Luca Savoldelli (2011) “Steady-state initialization of object-oriented thermo-fluid models by homotopy methods” In *Proceedings of the 8^h International Modelica Conference.* 86-96

Casella, Francesco (2012): On the Formulation of Steady-State Initialization Problems in Object-Oriented Models of Closed Thermo-Hydraulic System.

Cormen, Thomas H. and Charles E. Leiserson, and Ronald L. Rivest (1990) *Introduction to Algorithms.* MIT Press. ISBN 0-07-013143-0

Höbinger, Mathias and Martin Otter (2008): “PlanarMultiBody A Modelica Library for Planar Multi-Body Systems”. In: *Proceedings of 6^{th} International Modelica Conference* 549-546

Ochel, Lennart, and Bernhard Bachmann and Francesco Casella (2014): “Symbolic Initialization of Over-determined Higher-index Models” In: *Proceedings of the 10^h International Modelica Conference.* 1179 – 1187.

Olsson, Hans (editor) (2021): Modelica A Unified Object-Oriented Language for Systems Modeling Language Specification Version 3.5.
URL: https://specification.modelica.org/maint/3.5/MLS.pdf

Kuhn, Martin Raphael (2017): “Periodic Steady State Identification for use in Modelica based AC electrical system simulation”. In: *Proceedings of the 12^{th} International Modelica Conference.* 493 – 505.

Mattsson, Sven Erik, and Hans Olsson and Hilding Elmqvist (2000) “Dynamic Selection of States in Dymola”. In: *Modelica Workshop 2000.* 61 –67.

Mattsson, Sven Erik, and Hilding Elmqvist, and Martin Otter, and Hans Olsson (2002) “Initialization of hybrid differential-algebraic equations in Modelica 2.0”. In: *Proceedings of 2^{nd} International Modelica Conference.* 9-15.

Pantelides, Constantinos (1988) “The Consistent Initialization of Differential-Algebraic Systems” In *SIAM Journal on Scientific and Statistical Computing 9:2, pg 213-231.*

Sielemann, Michael, and Francesco Casella, and Martin Otter, and Christoph Clauß, and Jonas Eborn, and Sven Erik Mattson, and Hans Olsson (2011) “Robust Initialization of Differential-Algebraic Equations Using Homotopy” In *Proceedings of the 8^h International Modelica Conference.*

Sielemann, Michael (2012) “Probability-One Homotopy for Robust Initialization of Differential-Algebraic Equations” In *Proceedings of the 9^h International Modelica Conference.* 223-236.

Tay, Teng-Tiow, and Ivan Mareels, and John B. Moore (1998) *High Performance Control*. Birkhäuser Basel.

New Equation-based Method for Parameter and State Estimation

Luis Corona Mesa-Moles[1] Erik Henningsson[2] Daniel Bouskela[1]
Audrey Jardin[1] Hans Olsson[2]

[1]R&D, Electricité de France, France, {luis.corona-mesa-moles,daniel.bouskela,
audrey.jardin}@edf.fr
[2]Dassault Systèmes, Sweden, {erik.henningsson,hans.olsson}@3ds.com

Abstract

To get reliable simulation results from a Modelica model it is important to parametrize and initialize the model using the best estimate of the state of the system. Commonly, this state estimation is done by inverse calculation on a square system of equations requiring as many known values as states to be computed. In practice this constraint is an important limitation and, in addition, this method does not provide any information on the uncertainties or confidence level associated to the estimated state.

Taking advantage of the mathematical formulation of Modelica equations, this paper presents a new method to cope with the difficulties associated to the inverse calculation method. This approach adapts and extends the framework of data assimilation to provide a fully-integrated Modelica tool, which efficiently can handle every type of state estimation problem for static models. This method has been successfully tested with simple and complex Modelica models. Finally, the Modelica implementation of this technique allows to easily extend it to further applications.

Keywords: Modelica, parameter estimation, state estimation, model, data assimilation

1 Introduction

The safe and economically efficient operation of power plants and energy systems at large such as power grids or district heating and cooling networks rely on data that are used to assess the current state of the system and make predictions on the future states of the system at different timescales ranging from seconds to years. Finding the best estimation of the system state is important to diagnose functioning problems such as energy or efficiency losses and make the right decisions for predictive maintenance such as the best time to change a pump before it breaks.

The best estimation depends on the quality and the number of measurements. Unfortunately, raw data is always subject to uncertainties, and in general the number of states to be estimated far exceeds the number of available measurements. When dealing with a behavioral model (or digital twin) of the system, the term “states” refers to the initial values of the dynamic states and the values of the parameters, which are quantities that are not constrained by the model’s equations. Poor state estimation limits the predictive power of the model.

Modelica models must be initialized by giving values to all states. This is normally done by inverse calculation on a square system of equations that requires to provide as many known values as states to be computed. Because the number of states most often far exceeds the number of known variables, the user must make a choice between the states to be computed and the states to be manually initialized from assumptions. Therefore, there is no guarantee on the accuracy of the estimation, and therefore no guarantee on the accuracy of the predictions. Static or dynamic checks on the model do not provide any information on the consistency of the initial conditions as initial conditions are not constrained by the model’s equations (however, unphysical initial conditions often, but not always, lead to numerical divergence at simulation time). An answer to that problem is data assimilation.

Data assimilation is a set of techniques that combines statistical data on the measurements with a priori knowledge from the expert (the so-called background) and knowledge embedded in the model (the model’s equations) to provide the best estimate of the system initial state in the form of a mean value and confidence range for each estimated state. The quality and the accuracy of the estimated states will depend on the quality of the input data considered. Data assimilation uses all available information, that is, the more data that is provided by the user (mainly background and statistical data) the more accurate the result of the data assimilation will be. In case that this detailed information is not known by the user, uninformative prior or weakly informative prior (such as guess values) can be provided.

Data assimilation was initially developed for weather forecasting and proved to be essential for the accuracy of weather prediction, which is particularly difficult, the atmosphere being a chaotic system. It is therefore expected that this technique delivers good results on a larger timescale for power plants which are stable systems.

A previous work (Corona Mesa-Moles L. Argaud J.P., Jardin A., Benssy A., Dong Y, 2019) has shown that data assimilation can be used with a Modelica model to estimate the state of the secondary loop of a nuclear power plant. In this experiment, the data

DOI
10.3384/ecp21181129

assimilation algorithms were coded in Python[1] and the Modelica model used as a black box to provide numerical values to the iterations of the Python code. This method has some limitations such as lengthy calculations and the impossibility of estimating the values of boundary conditions when several operating points are considered. One reason for the time-consuming calculations is the need to compute numerical Jacobian matrices by multiple calls to the full model of the plant.

From Dymola's perspective the calibration option has already been considered. This option is primarily designed for the case where there are more measurements than states and parameters to estimate, and normally that there are time-serie(s) of measurements (Dassault Systèmes, 2021, Section 2). The implementation, which is found in the Dymola Design library, solves the non-linear least squares problem, using Levenberg–Marquardt, which uses Gauss–Newton internally (Fletcher, 1987, Sections 5.2 and 6.1).

This paper presents a solution that is better integrated with Modelica: it takes advantage of the knowledge of the mathematical form of the Modelica equations to compute analytical Jacobians and allows the coding of data assimilation algorithms directly in Modelica. It also paves the way to extracting, from the original Modelica model, the equations that are strictly necessary to perform the state estimation, thus reducing the size of the computations. These equations correspond to the observation operator that binds the measured values (the inputs of the calculation) to the states (the outputs of the calculations): after all, no equation is needed if the measured or observed variables are the same as the estimated ones (the observation operator is then the identity operator).

This paper is structured as follows. Section 2 presents the new equation-based method for parameter and state estimation. Section 3 describes how this method is implemented in Modelica and finally Section 4 presents the application of this new method on a complex Modelica model developed with the open source library ThermoSysPro.

2 The New Equation-based Method

2.1 Optimization problem

Data assimilation intends to combine different sources of information in order to estimate at best the true state of a system. The combination of different sources of information such as a physical model and observations can be understood as an optimization problem. There are different mathematical approaches to handle data assimilation problems, among which control theory (variational methods) and estimation theory (Kalman filter or optimal interpolation) can be cited. Variational methods define explicitly the optimization problem considering a cost function in which terms associated to the model and to the observations appear. Compared to other methods, the variational approach enables to handle easily non-linear models and combine different types of observations while keeping a very efficient computation time and accurate solution for the optimization problem.

The most well-known variational method in data assimilation is the so-called 3D-Var approach (see Asch M. et al., 2016 for more details). This classical approach can be extended to take into consideration the estimation of boundary conditions as well, and this is one of the main novelties presented in this article.

The objective of the method described in this article is to give the best estimate of the tuners x of a given model H considering a certain number of observations $y_{\mathrm{obs},k}$ (which can be measurements or design assumptions for example) obtained for a set of different boundary conditions p_k. In practice, considering the uncertainties related to the model P_{b}, to the boundary conditions $P_{\mathrm{b},k}$ and to the observations $P_{\mathrm{R},k}$ (all given in the form of covariance matrices) it is possible to compute the best estimate of the tuners x and of the boundary conditions p_k on the basis of an initial value x_{b} and $p_{\mathrm{b},k}$that correspond to the *a priori* knowledge of the state (the so-called background). The difference between the tuners and the boundary conditions is that boundary conditions vary from one operating point to another while tuners remain the same for all the operating points considered.

The cost function described in the previous paragraph, and corresponding to an extended version of the 3D-Var approach, is defined as follows:

$$\begin{aligned}\min_{x,\,p} J(x,p)\ \ \textbf{with}\, J(x,p) \\ &= n_{\mathrm{op}} \cdot \|x - x_{\mathrm{b}}\|^2_{P_{\mathrm{b}}^{-1}} \\ &+ \sum_{k=1}^{n_{\mathrm{op}}} (\left\|p_k - p_{\mathrm{b},k}\right\|^2_{P_{\mathrm{b},k}^{-1}} \\ &+ \left\|y_k - y_{\mathrm{obs},k}\right\|^2_{P_{\mathrm{R},k}^{-1}})\end{aligned} \tag{1}$$

Where:

- n_{op} are the total number of operating points;
- x are the tuners to be estimated;
- x_{b} are the tuners' background (*a priori* knowledge of the tuners to be estimated);
- p_k are the boundary conditions for operating point k;
- $p_{\mathrm{b},k}$ are the boundary conditions' background (*a priori* knowledge of the boundary conditions to be estimated);

[1] The Python codes were mainly based on the LGPL free distributed tool ADAO (Salome, 2018)

- $y_k = H(x, p_k)$ are the values computed by the model for operating point k and H is the Modelica model;
- $y_{\text{obs},k}$ are the observations (corresponding to the values computed by H);
- ${P_\text{b}}^{-1}$ is the tuner background error covariance matrix;
- ${P_{\text{b},k}}^{-1}$ is the boundary condition background error covariance matrix for operating point k
- ${P_{\text{R},k}}^{-1}$ is the observation error covariance matrix for operating point k.

It can be noted that weighted Euclidean norms have been used in the definition of cost function J, i.e. for a vector v, $\|v\|^2_{A^{-1}} = v^T A^{-1} v$, where A^{-1} is a positive definite matrix.

Solving this optimization problem gives the best estimate of tuner x (which is denoted by x_a) and of boundary conditions p_k (which is denoted by $p_{\text{a},k}$), these optimal solutions are referred to as the *analysis*.

The action of the model is captured by H, relating the observed variables to the tuners and boundary conditions ($y_k = H(x, p_k)$). However, for general Modelica models y_k is not explicitly given as a function of x and p_k. Instead, y_k is implicitly given by

$$G(x, p_k, y_k, z_k) = 0 \tag{2}$$

where G is the mathematical representation of the entire static Modelica model and z_k are all of the computed variables except y_k. Thus, in general we have $n_z \gg n_y$. As discussed in Section 1, G and z_k may, for certain applications, be substantially reduced in size by extracting from G only those equations that are necessary to link x and p_k to y_k. In this article we consider models with large algebraic loops comprising most of the model, where the connections between the tuners and observed variables are traced through these loops. Thus, a substantial reduction of G is not possible, see Section 3. Nonetheless, we consider such an extraction algorithm an interesting topic for future work.

When generating simulation code, a Modelica tool will perform a causalization, this means that the implicit equation $G(x, p_k, y_k, z_k,) = 0$ is transformed into the explicit equation $y_k = H(x, p_k)$, where z_k are treated as internal variables inside H. Thus, we may continue to use the notation and formulae as presented earlier in this section.

2.2 Computation of uncertainties

The data assimilation approach gives as well the possibility to compute the uncertainties associated to the parameters and boundary conditions that are estimated. The uncertainty associated to the analysis is given in the form of an error covariance matrix, the so-called analysis-error covariance matrix: $P_\text{a}$associated to x_a and $P_\text{a,k}$ associated to $p_{\text{a},k}$. They play the same role with respect to the analysis as P_b and ${P_{\text{b},k}}^{-1}$ with respect to the background.

In the case of the initial approach formulation of the data assimilation problem, it can be shown that, assuming that the model operator H can be approximated by a linear operator $\boldsymbol{H}$ in a neighborhood of the analysis, the analysis-error covariance matrix P_a is equal to the Hessian $\mathcal{H}$ of the cost function at x_a (see Tarantola A., 2005 and Bousserez N. et al., 2015). In such case, the Hessian of the cost function is given by:

$$\mathcal{H} = {P_\text{b}}^{-1} + \boldsymbol{H}^T {P_\text{R}}^{-1} \boldsymbol{H} = {P_\text{a}}^{-1} \tag{3}$$

For the final formulation of the data assimilation problem presented in Section 2.1, the Hessian of the cost function with respect to x and all the p_k can be computed by differentiating the final form of the cost function twice. The first differentiation of the cost function gives its gradient:

$$\nabla_x J = 2 \cdot n_\text{op} \cdot {P_\text{b}}^{-1}(x - x_\text{b}) + 2 \cdot \sum_{k=1}^{n_\text{op}} \left(\nabla_x {y_k}^T \cdot {P_{\text{R},k}}^{-1} \cdot \left(y_k - y_{\text{obs},k}\right)\right) \tag{4}$$

$$\nabla_{p_k} J = 2 \cdot n_\text{op} \cdot {P_{\text{b},k}}^{-1} \cdot \left(p_k - p_{\text{b},k}\right) + 2 \cdot \nabla_{p_k} {y_k}^T \cdot {P_{\text{R},k}}^{-1} \cdot \left(y_k - y_{\text{obs},k}\right), \forall\, k \tag{5}$$

From these expressions and in a neighborhood of the analysis (assuming that the model derivatives can be approximated by a linear model (the so-called linear tangent model), i.e. $\nabla_x y_k$ and $\nabla_{p_k} y_k$ can be considered as constant matrices) the second derivatives of the cost function can be easily computed as follows:

$$\mathcal{H}_x = 2\left(n_\text{op} \cdot {P_\text{b}}^{-1} + \sum_{k=1}^{n_\text{op}} \left(\nabla_x {y_k}^T \cdot {P_{\text{R},k}}^{-1} \cdot \nabla_x y_k\right)\right) \tag{6}$$

$$\mathcal{H}_{p,k} = 2\left(n_\text{op} \cdot {P_{\text{b},k}}^{-1} + \nabla_{p_k} {y_k}^T \cdot {P_{\text{R},k}}^{-1} \cdot \nabla_{p_k} y_k\right), \forall\, k \tag{7}$$

Considering the results presented in (Tarantola A., 2005 and Bousserez N. et al., 2015), the analysis-error covariance matrices for x and all the p_k can straightforwardly be derived from the previous expressions in order to compute the analysis uncertainties: $P_\text{a} = {\mathcal{H}_x}^{-1}$ and $P_{\text{a},k} = {\mathcal{H}_{p,k}}^{-1}$.

2.3 Solving method 1: Gradient descent

The gradient descent method is a well-known method to solve optimization problems. It is an iterative process based on the computation of the cost function J. It can be implemented in many different ways, and it can be used in the context of data assimilation to solve the optimization problem mentioned above.

When boundary conditions are considered in the optimization problem, this method can be described by the following algorithm:

```
- Initialization : $x_0 = x_\text{b}$ and $p_{0,k} = p_{\text{b},k}, \forall\, k$
- While $\|\nabla_x J\|$ or $\|\nabla_{p_k} J\|, \forall\, k > \varepsilon$ or $n \leq n_\text{max}$, do:
    o Compute $\nabla_x J$ and $\nabla_{p_k} J, \forall\, k$
    o Descent and update of $x$ and $p_k$
```

    - $x_{n+1} = x_n - \text{step} \cdot \nabla_x J$
    - $p_{n+1,k} = p_{n,k} - \text{step} \cdot \nabla_{p_k} J, \forall\, k$
  - $n = n + 1$

The explicit expressions of the gradients can be found in Section 2.2.

2.4 Solving method 2: Stationary point

The optimization problem presented in Section 2.1 can be interpreted as a minimization of a cost function J under the constraints that the model equations must be satisfied. In this case, a necessary condition to find the minimum of the cost function is that its gradient is equal to zero.

This solution can be derived from the following equations:

- $\nabla_x J = 0$
- $\nabla_{p_k} J = 0, \forall\, k$

where the model equations $y_k = H(x, p_k)$ are used in the computations of the above objective function gradients. The stationary point equations are typically solved using a Newton-type solver for optimization (Fletcher, 1987, Chapter 3). However, we do not need to take further steps here as writing the optimization problem all in Modelica allows for stating equations, not just algorithms. For details see Section 3.1.2.

2.5 Solving method 3: BFGS

As a third alternative, we consider the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm. It is similar to gradient descent, but modifies the search direction by a gradually improved approximation of the Hessian of J, costing no additional function or gradient evaluations. In our tests, we have used a golden-section line search to find the minimum of J in the descent direction. For details, we refer to (Fletcher, 1987, Sections 2.6 and 3.2).

3 Implementation

As discussed in Section 2.1, the data assimilation procedure involves a model H, observation data $y_{\text{obs},k}$, background data x_b, $p_{\text{b},k}$, and the uncertainties for the observation and background data P_R, P_b, $P_{\text{b},k}$ .These components are used to construct an optimization problem, here the 3D-Var formulation (1).

The above model, considered for data assimilation, is here referred to as *original model*. The observation, background, and uncertainty data will be collectively referred to as *user input.*

We identify the following requirements for a user-friendly implementation.

1. The original model should be easy to use both for data assimilation and normal simulation;
2. The optimization model should be separated from the original model and use general notation as in Section 2.1;
3. The Modelica extensions should be kept to a minimum allowing for easy modification and maintenance of the original and optimization models.

We use the expression *optimization model* rather than optimization algorithm since, defining the optimization problem in Modelica, allows us to use equations-based concepts, not just algorithms. The requirements suggest a separation between the original model, the optimization model, and the user input. Pending the eventual combination of them all in symbolic and numeric treatment.

In Sections 3.1 and 3.2 we describe the prototype implementation included in Dymola 2022 and 3DEXPERIENCE 2022x. All described features are enabled by:

```
    Hidden.Assimilation.Enable = true.
```

The flag and other names described below may change in future versions.

As a recurring example we consider the simple model that computes the mass flow rate of a fluid through a pipe. In Modelica, this example is described as follows:

```
model TSP_Pipe
  parameter Real K = 10;
  parameter Real delta_P = 2e5;
  parameter Real rho = 998.8404;
  Real Q;
  Real q;
  Real G[2];
equation
  G[1] = delta_P - K*rho*q*abs(q);
  G[2] = rho*q - Q;
  G = {0, 0};
end TSP_Pipe;
```

This model is equivalent to the one presented in (El Hefni B. and Bouskela D., 2017) and developed with ThermoSysPro (El Hefni B. and Bouskela D., 2019).

3.1 Definition of the data assimilation optimization problem in Modelica

3.1.1 Extended original model

To specify which variables in a Modelica model are to be considered for data assimilation, a new variable attribute is used. To this end, the Modelica built-in type `Real` has been extended with the attribute `assimilation`, which is a record of type

```
record Assimilation
  StringType iotype;
  RealType data[:];
  RealType uncertainty[:];
end Assimilation;
```

where `StringType` and `RealType` are the Modelica primitives for strings and floating-point numbers, respectively (MLSv35 (Modelica Association 2021), Section 4.8). The string `iotype` designates a `Real` variable as a tuner x (`"Tuner"`), boundary condition p_k (`"BC"`), or observed variable y_k `"Observed"`. In the elements `data` and `uncertainty`, background or observed data and their respective uncertainties can be specified. For tuners these two elements are scalars, for boundary conditions and observed variables they are vectors of size n_{op}, the number of operating points. Note that, the elements `data` and `uncertainty` can also be modified later, see the upcoming section.

Consider again the pipe model `TSP_Pipe` introduced in Section 3. As to not modify the original model we may extend from it and specify the assimilation attributes as modifiers.

```
model TSP_Pipe_Assimilation
  extends TSP_Pipe(
    K(assimilation(
      iotype="Tuner",
      data=950.0,
      uncertainty=100000000)),
    delta_P(assimilation(
      iotype="Tuner",
      data=2.0e5,
      uncertainty=100000000)),
    rho(assimilation(
      iotype="BC",
      data={998.8, 995.0},
      uncertainty={1, 1})),
    Q(assimilation(
      iotype="Observed",
      data={447.0, 450.0},
      uncertainty={1, 1})));
end TSP_Pipe_Assimilation;
```

Two operating points are considered with relatively high confidence in the observations and boundary conditions. There are more parameters to estimate than observations.

3.1.2 Interface for the optimization model

With the original model extended to include the relevant user input, it remains to formulate the optimization problem and equations to find an optimum. To allow for general optimization models, Dymola automatically generates a library `AssimilationPrototype`, containing the wrapper model `AssimilationModel`. The interface consists of

```
input Real x[nx] "Tuners";
input Real p[nop, np] "BCs";
output Real y[nop, ny] "Observed";
output Real dydx[nop, ny, nx] "Jacobians";
output Real dydp[nop, ny, np] "Jacobians";
```

in combination with parameters for all of the measurement data, background data, and uncertainties. The dependency $y_k = H(x, p_k)$ imposed by the original model (Section 2.1) is handled by the wrapper, cf. Section 3.2.

To exemplify the usage of this interface we construct a simple and equation-based optimization model for the 3D-Var optimization problem (1), with gradients as derived in Section 2.2.

```
partial model DA_3DVar
  AssimilationPrototype.AssimilationModel
    mod(x=x, p=p);
  Real x[mod.nx];
  Real p[mod.nop, mod.np];

  Real dJdx[mod.nx] = 2*mod.nop*
    (x - mod.x_bg)*mod.P_b_inv +
    sum(2*(mod.y[k,:] - mod.y_obs[k,:])*
      mod.P_R_inv[k,:,:]*mod.dydx[k,:,:]
      for k in 1:mod.nop);

  Real dJdp[mod.nop, mod.np] =
    {2*(p[k,:] - mod.p_bg[k,:])*
    mod.P_bk_inv[k,:,:] +
    2*(mod.y[k,:] - mod.y_obs[k,:])*
    mod.P_R_inv[k,:,:]*mod.dydp[k,:,:]
    for k in 1:mod.nop};
end DA_3DVar;
```

The `AssimilationModel` component `mod` is used to access the dimensions of the data assimilation problem, the parameters containing all user input, and the co-variance matrices. The inputs x and p_k are bound to the local unknowns with the same name. The observed variable y_k together with the Jacobians $\nabla_x y_k$ and $\nabla_{p_k} y_k$ are computed in the `AssimilationModel`. Using the Jacobians, the objective gradients $\nabla_x J$ and $\nabla_{p_k} J$ can also be computed, cf. Equations (4, 5). The model is partial as there are $n_x + n_{\mathrm{op}} \cdot n_p$ more variables than equations; an optimality condition has to be enforced.

```
model StationaryPoint
  extends DA_3DVar;
equation
  dJdx = zeros(mod.nx);
  dJdp = zeros(mod.nop, mod.np);
  annotation(experiment(StopTime=0));
end StationaryPoint;
```

Here, we reap the full benefit of equation-based modelling as we simply set the derivatives to zero to find a stationary point, cf. Section 2.4. Dymola will do the rest and employ its Newton-type nonlinear system solver as a Newton method for optimization; there is no need to write an optimization algorithm.

Finally, a special top-level annotation is provided to point to the original model to assimilate.

```
model StationaryPoint_TSP_Pipe_Assimilation
  extends StationaryPoint;
  annotation(assimilationModel =
    "TSP_Pipe_Assimilation");
end StationaryPoint_TSP_Pipe_Assimilation;
```

When this model is translated, Dymola also translates the original model `TSP_Pipe_Assimilation` and then populates the wrapper model `AssimilationModel` with the specifics of the original model. The parameters containing the user input have default values giving by the `assimilation` attributes in the original model. These parameters can also be modified from the optimization model, indeed even after translation in Dymola's Variable Browser.

Translating `StationaryPoint_TSP_Pipe_Assimilation` we get a single nonlinear system to be solved for the two tuners and the two operating points of the single boundary condition. When performing the static simulation, the system is solved using 19 iterations and one Jacobian computation. Note that the Jacobian of `simulation.nonlinear[1]` is the Hessian of J. The results are as follows: `x[1] = K = 991.2` and `x[2] = delta_P = 199999.8`. For the boundary condition `rho`, we get for the respective operating points `p[1,1] = 998.4` and `p[2,1] = 995.4`. Note how the low uncertainty for the boundary conditions results in values close to their background data.

Other optimization models may use the time as an iteration variable. That is, let all variables be discrete, in particular the iterates x and p_k. Then, the `sample` operator can be used to perform one iteration at each sample. For example, a simple Gradient Descent algorithm can be implemented as follows.

```
 when sample(1,1) then
  x = pre(x_new);
  p = pre(p_new);
 end when;
 x_new = x - dJdx;
 p_new = p - dJdp;
```

The iterations may be stopped at convergence using the `terminate` operator. For more complex optimization models the `AssimilationModel` wrapper also offers auxiliary functions `ComputeOutput` and `ComputeGradient` to be used in algorithms. For example BFGS can use the former to update y_k and eventually J when performing line-search.

3.1.3 Robustness of the optimization model

When performing data assimilation under the 3D-Var formulation (1), an optimization algorithm can use the available background and observed data to improve robustness of the algorithm. For example, the solution is expected to be close to the background data, so this data can be used for starting guesses, nominals, and other scaling. The variables in the `AssimilationModel` wrapper are automatically assigned these attributes.

To additionally improve the robustness of the `StationaryPoint` optimization model, homotopy can be used. Again, the 3D-Var formulation lends itself to a natural choice: When the uncertainties for the observed variables y_k tends to infinity, the 3D-Var formulation breaks down to the simple assignments $x = x_\mathrm{b}$ and $p_k = p_{\mathrm{b},k}$. Similarly, the higher the uncertainties are for y_k the easier the system `simulation.nonlinear[1]` is to solve as the influence of the model is limited. Based on this we propose the following homotopy variant of `StationaryPoint`.

```
model StationaryPoint_Homotopy
 extends StationaryPoint(mod(
   y_obs_w = homotopy(mod.y_obs_w_const,
     1.0e4 * tuner_bg_w_max *
     ones(mod.nop, mod.ny))));
 constant Real tuner_bg_w_max =
  max(max(mod.x_bg_w_const),
    max(max(mod.p_bg_w_const)));
end StationaryPoint_Homotopy;
```

For the simple model, the uncertainties `y_obs_w` for y_k are set to 10^4 times the largest uncertainty of the tuners and background variables. During the homotopy process `y_obs_w` is successively decreased to attain its actual values. Here, the `_const` variables, also provided by the wrapper, are constant versions of the corresponding user input parameters.

3.2 Dymola implementation, back-end

The full optimization model consists of three parts:

- The transformed original model;
- The computation of the Jacobians $\nabla_x y_k$ and $\nabla_{p_k} y_k$;
- The optimization model.

In the following subsections we build up the optimization problem, item by item.

3.2.1 Transformation of the original model

The goal of data assimilation is to determine values for parameters (including initial values for states) in the original model. Thus, only Modelica parameters can be designated as assimilation tuners (`iotype = ″Tuner″`) or boundary conditions (`iotype = ″BC″`). To ease the presentation, we also enforce that all observed variables (`iotype = ″Observed″`) must be computed variables (non-parameters) in the original model. That is, we disregard cases where a parameter in the original model has both measurement data and background data. The generalization to these cases merely means including such parameters both in the x (or p_k) vector and in the y_k vector.

To prepare the original model for data assimilation, Dymola automatically removes the bindings on the parameters to be assimilated. These parameters are afterwards transformed to inputs and the observed variables are transformed to outputs, matching the interface of the `AssimilationModel` wrapper described in Section 3.1.2. As the original model is assumed to be a normal simulation model, we may assume that it is determined (square). These transformations keep this determinacy.

3.2.2 Computation of the Jacobians

In the previous section, we saw how the original model readily can be transformed to fit into the structure of the `AssimilationModel` wrapper. Assuming known values for the inputs, we get a square transformed original model, here considered in its implicit form $G = 0$, cf. Section 3.1.1. After achieving a static simulation result for each operating point, it is straight-forward to formulate the adjoint equations for the Jacobians $\nabla_x y_k$ and $\nabla_{p_k} y_k$ around these solutions

$$0 = \nabla_{x_i} G + \nabla_{y_k} G \cdot \nabla_{x_i} y_k + \nabla_{z_k} G \cdot \nabla_{x_i} z_k \qquad (8)$$

$$0 = \nabla_{p_{k,i}} G + \nabla_{y_k} G \cdot \nabla_{p_{k,i}} y_k + \nabla_{z_k} G \cdot \nabla_{p_{k,i}} z_k \qquad (9)$$

for $k = 1, \dots, n_{\mathrm{op}}$ and $i = 1, \dots, n_{\mathrm{x}}$ (or n_{p} respectively). These are $n_x \cdot n_{\mathrm{op}} + n_p \cdot n_{\mathrm{op}}$ linear systems of equations of size $n_G = n_y + n_z$. If we compute the partial derivatives of G analytically once and for all, then it is quick work to set up the Equations (8, 9) at each pair (i, k). However, note that G is the entire static model (or at least a major part of it, cf. Section 2.1). In consequence z_k and y_k are all (or most) of the computed variables. Thus, the size of each of the Equations (8, 9) quickly becomes large. Additionally, the number of introduced variables $n_{\mathrm{op}} \cdot (n_y + n_z) \cdot (n_x + n_p)$ becomes unfeasibly large even for medium-sized models. For these reasons we chose not to use the adjoint equations for the Jacobian computations, notwithstanding recognition of several additional optimization that could have been done to Equations (8, 9).

Instead, we consider the built-in chain rule in Dymola that is used when computing analytic Jacobians for dynamic simulation and input-output-Jacobians for FMUs. Indeed, the transformed original model has already been endowed with inputs (x and p_k) and outputs (y_k) preparing it for the application of the chain rule.

The chain rule in Dymola internalizes all of the auxiliary variables z_k (compare $y_k = H(x, p_k)$) and caches all of the partial derivatives. As most equations in a typical Modelica model are simple, the partial derivatives in turn are also mostly simple. There are important exceptions to this rule. Most notably, we have assumed that the model has a large algebraic loop, cf. Section 2.1. Thus, for most partial derivatives of y_k with respect to most x or p_k, the chain rule has to run through the algebraic loop. To this end, the inverse of the system Jacobian for the loop is needed. Computing it is costly but can be done quite efficiently in Dymola by the help of tearing and caching.

Dymola's chain rule gives us even more for free: The dependencies for the partial derivatives are traced between each y_k and x or p_k (caching any new information). This means that only the equations that are actually needed for the partial derivative computations are extracted and used.

3.2.3 Efficient treatment of several operating points

The analytical expressions for the partial derivatives $\nabla_{p_k} y_k$ and $\nabla_{p_l} y_l$ for $k, \ell = 1, \dots, n_{\mathrm{op}}$ are identical. Therefore, the chain rule does not need to be applied to each operating point. This fact is reflected in the transformed original model, where each boundary conditions p_k and observed variable y_k only corresponds to a single input or output, respectively.

Instead, we handle the different operating points in the `AssimilationModel` wrapper. Namely, in the array `original[nop]`, where one copy of the original model is instantiated per operating point. The wrapper then takes care to broadcast the values of the input matrix `p[nop,np]` to the correct instances of the original model, cf. Section 3.1.2. Similarly, the wrapper collects the observed variables and Jacobians from the instances into the wrapper outputs.

We conclude that the Jacobian computations proposed here only extend the original model with $n_y \cdot (n_x + n_p)$ additional scalar variables, constituting $\nabla_x y_k$ and $\nabla_{p_k} y_k$. Noting that typically $n_z \gg n_y$, this is a crucial improvement over the straight-forward application of the adjoint equations (8, 9).

3.2.4 Causality of the data assimilation optimization model

Up to this point, the causality of the original model has been preserved; the transformed model is evaluated first, after which the Jacobians are computed. Optimization models using time and sample to iterate does not change this causality. With the proper choices `x` and `p` as iterates, their old values `pre(x)` and `pre(p)` are inputs at each iteration and can be sent to the transformed original model for computation of observed variables and Jacobians, eventually leading to an update of the iterates.

However, recall that our aim is to allow for the full power of equation-based modelling when writing optimization models for data assimilation. For example, the stationary point models presented in Sections 3.1.2 and 3.1.3 take advantage of this feature. The equations

```
dJdx = zeros(mod.nx);
dJdp = zeros(mod.nop, mod.np);
```

involve all computed variables of the optimization problem. In particular, the unknowns are the tuners and boundary conditions, the computed variables of the original model, the Jacobians and the gradients. The equations are those of the original model, their derivatives with respect to the tuners and boundary conditions, and the stationary point equations. We return to the `TSP_Pipe` example for an illustration

$$0 = G_1 = \Delta P - \rho K q |q|,$$
$$0 = G_2 = \rho q - Q,$$

with $x = (K, \Delta P)$ as tuners and $y = Q = Q(K, \Delta P)$ as observed variable, leaving $z = q = q(K, \Delta P)$ as an auxiliary variable (and disregarding boundary conditions). The stationary point optimization model for `TSP_Pipe` has the following incidence.

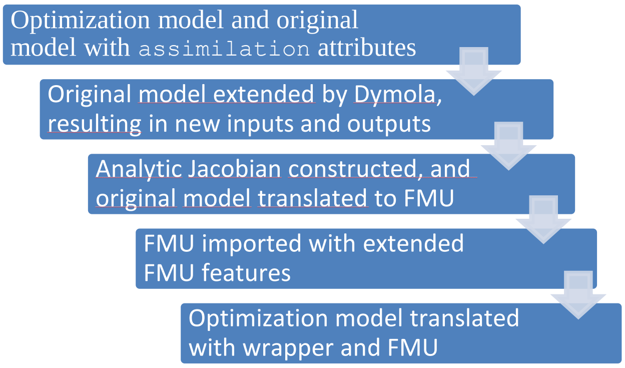
	K	ΔP	Q	q	$\frac{\partial Q}{\partial K}$	$\frac{\partial q}{\partial K}$	$\frac{\partial Q}{\partial \Delta P}$	$\frac{\partial q}{\partial \Delta P}$
G_1	*	*		*				
G_2			*	*				
$\mathrm{d}G_1/\mathrm{d}K$	*			*		*		
$\mathrm{d}G_2/\mathrm{d}K$					*	*		
$\mathrm{d}G_1/\mathrm{d}\Delta P$	*			*				*
$\mathrm{d}G_2/\mathrm{d}\Delta P$							*	*
$\mathrm{d}J/\mathrm{d}K$	*		*		*			
$\mathrm{d}J/\mathrm{d}\Delta P$		*	*				*	

As the example suggests, a partitioning can in general not be made to solve the stationary point optimization model in sequence. Rather, it must be considered at once as a (nonlinear) system of equation. The tuners and boundary conditions constitute a natural choice for iteration (tearing) variables as, when they are eliminated, the original model, the Jacobians, and the stationary point equations can be computed in sequence, where the latter are residual equations.

Commonly, and here by assumption (Section 2.1), the original model contains algebraic loops. Either these loops can be handled in the same nonlinear system as the optimization system, or they can be nested inside the optimization system. In the former case, the tuner and boundary condition iteration variables are mixed with the iteration variables for the loops in the original model. In the latter case, the loops of the original model are considered as blocks inside the optimization loop, and are solved in full each outer iteration. Compare DAE mode versus ODE mode for dynamic simulation (Braun, Casella and Bachmann, 2017; Henningsson, Olsson and Vanfretti, 2019). To keep the system of equations as small as possible we here choose the latter alternative. Additionally, it is more in line with the default choice of ODE mode for dynamic simulations in Dymola. A deeper investigation of these two alternatives may be an interesting topic for future work.

3.2.5 Synthesis

We summarize with a flowchart of the steps taken by Dymola when translating the full data assimilation problem to generate simulation code. To wrap the original model with Jacobian computations we chose to use (an extension of) FMI 2.0 for Model Exchange, which comes ready with an input-output interface and analytic Jacobian computations in Dymola.


Optimization model and original model with assimilation attributes
Original model extended by Dymola, resulting in new inputs and outputs
Analytic Jacobian constructed, and original model translated to FMU
FMU imported with extended FMU features
Optimization model translated with wrapper and FMU

In the last step, the causality of the optimization model is established and an (outer) nonlinear system is constructed if so needed. All the steps are done automatically by Dymola and the user does not need to be concerned about casualization questions. That is, the assimilation prototype uses the equation-based paradigm, as any other Modelica model.

In conclusion, the analytic Jacobian only need to be constructed once, which is done when translating the extended original model. This Jacobian can then be cheaply evaluated several times throughout the optimization procedure. In contrast, consider the traditional gradient-based approaches, e.g., those discussed in the Introduction, there a numeric Jacobian has to be constructed for each operating point during each optimization iteration.

4 Experimentation results on a more complex example

A complex ThermoSysPro model of the secondary loop of a pressurized water reactor is retained for this experimentation. It is the same model as the one presented in (Corona Mesa-Moles L. Argaud J.P., Jardin A., Benssy A., Dong Y, 2019). This model, presented in Figure 1, has over 12000 equations and it is used to compute the nominal operation point of the secondary loop. It is mainly composed of the following subsystems:

- A turbogenerator set made of high-pressure and low-pressure turbines and one generator;
- Two sets of moisture separator reheaters;
- One condenser;
- One feedwater tank and gas stripper system;
- Two turbine-driven feedwater pumps;
- Low and high pressure feedwater headers.

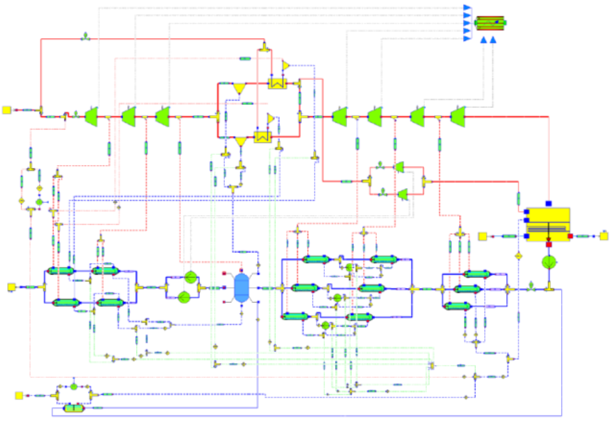


Figure 1. Model of the secondary loop of a 1300MW pressurized water reactor

In order to assess the correct implementation of the data assimilation method in Modelica, a twin experiment is considered. In such type of experiments, the observed variables used to perform data assimilation come from the simulation itself for a given state (the so-called reference state and obtained with a given value of tuners and boundary conditions) and the goal is to evaluate how close to this state the optimal state estimated through data assimilation is. The BFGS implementation is the method used for this experimentation.

In this twin experimentation, a state defined by three tuners and two boundary conditions for two different operating points is considered. This information, including the observed variables related to this state are given in Table 1.

It can be noted that in this experimentation the number of observed variables (3) is not be enough to correctly calibrate both tuners (3) and boundary conditions (2) using the usual method using square system. Data assimilation offers a general approach to deal efficiently with this type of calibration scenarios.

Table 1. Generation of the reference state

Quantity		Value	
		Op. Point 1	Op. Point 2
Tuners[2] x	1	2373.13	-
	2	2373.13	-
	3	405.762	-
Boundary conditions[3] p	1	3802.63	3820
	2	51.42	48
Observed variables[4] y	1	22753.38	223814.23
	2	22753.38	223814.23
	3	382447.47	383190.88

4.1 Scenarios considered

Two scenarios are considered to assess the correct implementation of the data assimilation technique in Modelica. Since one of the main novelties of this implementation is to take into consideration boundary conditions (BCs) in the state estimation problem, the two scenarios differ only on the *a priori* uncertainties associated to these boundary conditions.

The data assimilation inputs for Scenario 1 are given in Table 2. For observed variables and as a twin experimentation is performed, the values provided in Table 1 are kept with a low uncertainty associated (10^{-1} for each of them).

Table 2. Data assimilation inputs for Scenario 1

Quantity			Background	Uncertainty[5]
Tuners x	1		2450	10^{7}
	2		2320	10^{7}
	3		500	10^{7}
BCs p	1	Op. Point 1	3802.63	10^{-2}
		Op. Point 2	3820	10^{-2}
	2	Op. Point 1	51.42	10^{-4}
		Op. Point 2	48	10^{-4}

For Scenario 1, the uncertainties related to the tuners are much larger than the ones considered for the boundary conditions, the goal is that data assimilation will mainly adjust the values of the tuners. Therefore, it is expected for the adjusted value of the boundary conditions to be very close to the background values.

Scenario 2 is identical to Scenario 1 excepted for the uncertainty related to boundary conditions. In this scenario, an uncertainty of 10^{1} is considered for all boundary conditions. Compared to Scenario 1, it is expected to obtain different values for the optimal state and in particular for the boundary conditions.

4.2 Results

The results obtained for Scenario 1 are given in Table 3. In Table 3 the optimal values for tuners and boundary conditions within their associated uncertainties are detailed.

[2] Tuners considered in this example correspond to heat transfer coefficients of different heat exchangers.

[3] Boundary conditions considered in this example are the total thermal power extracted from the steam generators and the mass flow rate of the cooling water.

[4] Observed variables considered in this example correspond to enthalpies taken at different locations of the secondary loop.

[5] Diagonal values of the background error covariance matrices (P_{b} and $P_{\mathrm{b},k}, \forall\ k$)

Table 3. Optimal state estimation for Scenarios 1 and 2

Quantity			Analysis Scenario 1	Analysis Scenario 2
Tuners x	1		2373.1304	2386.46
	2		2373.1306	2386.46
	3		405.76154	417.57
BCs p	1	Op. Point 1	3802.6245	3828.46
		Op. Point 2	3819.9993	3846.12
	2	Op. Point 1	51.425	54.10
		Op. Point 2	47.99998	50.29

For Scenario 1, as expected due to their low background uncertainty, the values of the analysis for boundary conditions are very similar to the ones given as background (see Table 2). This is not the case for tuners for which the background uncertainty was much larger. As a consequence, the associated analysis values differ from the background ones and as expected the optimal values of the tuners correspond to the ones used to generate the reference state (see Table 2). This result shows that the data assimilation procedure is correctly implemented.

For Scenario 2, the analysis values for both tuners and boundary conditions are different from the initial background value. The higher uncertainty given to the boundary conditions is responsible for this result. For the tuners it is not surprising to find a different value from the one used to generate the reference state: this is an adjustment made as a consequence of the new analysis values for boundary conditions.

The computed uncertainties[6] for the analysis values of tuners and boundary conditions are low and they are very similar for both scenarios. For tuners 1 and 2 the uncertainty is around 10^{-4}, for tuner 3 the corresponding uncertainty is even lower, around 10^{-6}. For boundary conditions, the uncertainty is around 10^{-5} for the first one and around 10^{-7} for the second one.

With respect to the observed variables, the results are as well satisfactory. For both scenarios the observed variables corresponding to the optimal state (optimal values for tuners and boundary conditions) are extremely close to the observed variables computed from the reference state. In order to evaluate the quality of this adjustment, one can examine the evolution of the cost function (as defined in Section 2.1): from $6.60\text{x}10^{8}$ to 0.006 for Scenario 1 and from $6.60\text{x}10^{8}$ to 136.9 for Scenario 2. These results confirm the good implementation of the data assimilation procedure in Modelica.

In addition to the numerical results, it is interesting to point out some results with respect to the computation time. With the current non-optimized implementation of the BFGS algorithm and on a standard machine, Dymola requires 15.1 seconds per iteration in average. This time should be compared to the 26.6 seconds per iteration on average for the ADAO implementation of data assimilation. The time reduction per iteration obtained with this new equation based approach is therefore already considerable. Practically all the time required for the simulation is spent on model evaluations; reducing the time required for this task appears therefore as a good solution to reduce the time required to perform the data assimilation procedure (for example extracting only the model equations that are necessary to complete the calculations). In addition, the stopping criteria retained for the optimization algorithm may have an important impact on the total number of iterations (and therefore on the total time) required to solve the optimization problem.

5 Conclusion and perspectives

5.1 Conclusion

Based on the mathematical form of the Modelica equations, this paper presents a new method for parameter and state estimation of Modelica models. This method considers the problem of state estimation as an optimization problem and it has been adapted from the data assimilation framework.

Traditionally, this task is performed in Modelica by inverse calculation on a square system of equations which requires that the user provides as many known values as states to be computed. In practice this is an important limitation to state estimation since it is very rare to have the same number of known values as states to be estimated. The new method presented in this paper enables to efficiently handle non-square problems which are the most frequent ones.

Integrating this approach directly in a Modelica tool allows to use the analytic expressions of the Jacobians that are necessary to solve the optimization problem. The time necessary for computation can therefore be reduced compared to other methods in which numerical Jacobians have to be computed. In addition, with this approach it is possible to compute the uncertainties of the final estimated state (making it possible to specify the uncertainties related to tuners and/or boundary conditions for example). This information is an important tool for the user to evaluate the adequacy of the estimated state.

The prototype implementation in Dymola 2022 and 3DEXPERIENCE 2022x of this new method has been tested successfully with different simple and complex

[6] Diagonal values of the analysis error covariance matrices (P_a and $P_{\text{a},k}, \forall\ k$)

Modelica models such as the model of a secondary loop of a pressurized water reactor.

5.2 Future work

As we saw in Section 3.2.2, when computing the Jacobians, we only use those equations in the original model that are actually needed. However, for the function evaluations, i.e., the computations of y_k, we still consider the full model. The data assimilation procedure therefore needs to assume that the model simulates successfully around the solution of the optimization problem. This is a reasonable assumption when the solution is close to the background data, where it, in turn, is reasonable to expect that the model is well behaved. On the other hand, data assimilation techniques may also help in the initialization of failing models. For example, by starting with a small amount of assimilation variables, constituting only a part of the model, and then successively adding more variables. Such a procedure requires that only the relevant equations are extracted also for function evaluations. Additionally, the model topology must allow it, i.e., the model must not entirely be made up by an algebraic loop. We consider this an interesting topic for future work, as the challenging problem of robust initialization is one of the major obstacles in contemporary Modelica applications.

To ease the presentation, we have only discussed optimization with no lower or upper bounds on the tuners and boundary conditions. However, the presented optimization models can easily be extended to take into account such constraints. To better support this the `assimilation` attribute may be extended in future Dymola releases to allow specification of bounds directly in the original model.

Finally, we have limited our presentation and implementation to focus on the data assimilation problem. However, as the objective function and optimization models are written all in Modelica the techniques can be extended to more general optimization problems.

Acknowledgements

This work was funded and supported by BPI France through the French FUI ModeliScale research project.

References

Asch M., M. Bocquet, M. Nodet (2016), *Data Assimilation - Methods, Algorithms and Applications*, SIAM.

Bousserez, N., D.K. Henze, A. Perkins, K.W. Bowman, M. Lee, J. Liu, F. Deng and D.B.A. Jones (2015). “Improved analysis-error covariance matrix for high-dimensional variational inversions: application to source estimation using a 3D atmospheric transport model”. In: *Q.J.R. Meteorol. Soc.,* 141: 1906-1921. https://doi.org/10.1002/qj.2495

Braun W., F. Casella and F. Bachmann (2017). “Solving large-scale Modelica models: New approaches and experimental results using OpenModelica”. In *Proceedings of the 12th International Modelica Conference,* pages 557–563. Linköping University Electronic Press,.

Corona Mesa-Moles L. J.P. Argaud, A. Jardin, A. Benssy, Y. Dong (2019). “Robust Calibration of Complex ThermosysPro Models using Data Assimilation Techniques: Application on the Secondary Loop of a Pressurized Water Reactor”. In: *Proceedings of the 13th International Modelica Conference,* pages 553-560. Linköping University Electronic Press.

Dassault Systèmes (2021). *Dymola 2022: Dymola, Dynamic Modeling Laboratory, User Manual 2A: Model Development Tools.* Dassault Systèmes AB, Lund, Sweden.

El Hefni B. and D. Bouskela (2017). “Modeling and simulation of a complex ThermoSysPro model with OpenModelica – Dynamic Modeling of a combined power plant”. In: *Proceedings of the 12th International Modelica Conference*, May 15-17, Prague, Czech Republic.

El Hefni, B. and D. Bouskela (2019). *Modeling and Simulation of ThermalPower Plants with ThermoSysPro - A Theoretical Introduction and a Practical Guide*. Springer.

Fletcher, R. (1987). *Practical methods of optimization*. 2nd edn. New York: John Wiley & Sons.

Henningsson E., H. Olsson and L. Vanfretti (2019). “DAE Solvers for Large-Scale Hybrid Models”. In: *Proceedings of the 13th International Modelica Conference*, pages 491–502. Linköping University Electronic Press.

Modelica Association (2021-02). Modelica – A Unified ObjectOriented Language for Systems Modeling. Language Specification Version 3.5. Tech. rep. Linköping: Modelica Association. URL: https://specification.modelica.org/maint/3.5/MLS.html.

SALOME The Open Source Integration Platform for Numerical Simulation, accessed 2021-08-19, http://www.salome-platform.org/

Tarantola A. (2005). *Inverse Problem Theory and Methods for Model Parameter Estimation*. SIAM: Philadelphia, PA.

Efficient Parameterization of Modelica Models

Thomas Beutlich[1] Dietmar Winkler[2]

[1]Germany, modelica@tbeu.de
[2]University of South-Eastern Norway, dietmar.winkler@usn.no

Abstract

This article presents the different approaches and use cases for efficient parameterization of Modelica models by means of external data resources. The main motivation is to improve the overall quality, testability and reusability of Modelica application models (both on component and system level) by a separation of the behavioral implementation from its actual design parameters. The Modelica libraries `ExternData` and `ModelicaTableAdditions` are freely available to support library developers and vendors in their ambitions to offer clean and dedicated interfaces for the parameterization of the application models and to benefit from a large variability of commonly used file types, such as CSV, Excel, HDF, JSON, MATLAB MAT, XML or even domain-specific file types such as for tire properties or weather data.

Keywords: parameterization, external data resources, Modelica external function interface, SSP

1 Introduction

The separation of the design parameters from Modelica application models was already discussed within the MA (Modelica Association)[1] about 15 years ago. Tiller (2005) developed an in-house library `DataRetrieval`, that featured a generic approach applicable for different file formats or data bases. Supported file formats included for example XML (eXtensible Markup Language), HDF (Hierarchical Data Format) and MATLAB MAT. There even have been early ideas for the standardization of the appropriate interfaces and XPath query expressions. Similarly, Köhler and Banerjee (2005) presented an in-house library `ZFlib` based on simple ASCII text files for a generic parameterization of transmission models. This library was later extended by Kellner et al. (2006) to also support target platforms without a file system. Reisenbichler et al. (2006) bewailed that the XML technology had not yet established as a standardized concept for the parameterization of Modelica application models. They again proposed to use XML as file format for external data resources – being a standardized and widely accepted language with significant tool support for data processing. Their Dymola[2] library also featured the full power of the XPath query expressions and data processing capabilities.

The topic was raised again for the MSL (Modelica Standard Library)[3] without greater reception in 2008[4]. Therein it was mentioned, that with the current concept of implementing the data access by the Modelica external function interface, the library vendors and users are responsible to instrument the Modelica code to consider parameterization (described as *pull*-principle). However, with a *push*-principle this responsibility could be moved to the tool vendors, and as such library users could benefit from a greater reusability and flexibility of the layered parameterization.

When the MA project SSP (System Structure and Parameterization of Components for Virtual System Design)[5] was initiated in 2014, only the parameterization of networks of FMUs (Functional Mock-up Units)[6] was considered. Even though the SSP standard 1.0 misses support for array parameters, it was not yet contemplated to apply it as layered standard for the parameterization of Modelica models.

With no standardized interface available, Modelica users depending on external data resources either still need to write their own utility libraries or have to depend on proprietary, tool-specific features (e.g., the data base interface of SimulationX[7]) or commercially available libraries such as `Modelon.DataAccess` from Modelon[8].

The parameterization of Modelica models can be differentiated by the following usage scenarios.

- Property parameters are constant during a transient simulation. They are non-structural parameters, i.e., a translated simulation model can be reused with changed parameters. Examples are geometry dimensions (e.g., tire diameter), material constants (e.g., electrical resistance) or ambient conditions (e.g., ambient pressure or gravitational acceleration). They can be of `Real`, `Integer`, `Boolean` or `String` type and either be scalar or of one/two-dimensional array kind (e.g., consumption map or road excitation map).

[1] Modelica Association, https://modelica.org

[2] Dassault Systèmes, https://www.3ds.com

[3] MA project "Libraries", https://doc.modelica.org

[4] MSL issue #115, https://github.com/modelica/ModelicaStandardLibrary/issues/115

[5] MA project "System Structure and Parameterization", https://ssp-standard.org

[6] MA project "Functional Mock-up Interface", https://fmi-standard.org

[7] SimulationX by ESI, https://www.simulationx.com

[8] Modelon, https://www.modelon.com

DOI
10.3384/ecp21181141

- Stimulation parameters can be considered as time-driven inputs for a transient simulation and can be modeled by one-dimensional look-up tables. Examples are the environmental conditions such as weather.
- Structural parameters have influence on the overall system topology and thus on the dimension of the system of equations. They need to be constant during a transient simulation, but any change requires a new translation of the Modelica model. Special care needs to be taken if structural parameters are read from external data resources.

The Modelica libraries `ExternData`[9] and `ModelicaTableAdditions`[10] are available as open-source Modelica packages under the permissive BSD-2-Clause License. Both libraries can be directly obtained from GitHub or via the Modelica `impact` package manager (Tiller and Winkler 2014; Tiller and Winkler 2015).

- `ExternData` supports the user in reading properties or structural parameters from various file types of external data resources. Data access from CSV (Comma Separated Values), INI, JSON (JavaScript Object Notation), MATLAB MAT (including HDF), SSV (System Structure Parameter Values), TIR (Tire Properties), Excel XLS/XLSX and XML files is implemented.
- `ModelicaTableAdditions` is an extension of the Modelica Standard Tables (Beutlich, Kurzbach, and Schnabel 2014) with support for more file types beside Dymola MOS[11] and MATLAB MAT. Its blocks can be utilized to also model stimulation parameterization or look-up tables from CSV, EPW (EnergyPlus Weather)[12] or JSON files and work as a drop-in replacement for the Modelica Standard Tables of the MSL.

There exists the use case to reuse the time series stored in the result data of one simulation as stimulation parameterization for subsequent simulations. Whereas Pfeiffer, Bausch-Gall, and Otter (2012) designed an HDF based file format, Tiller and Harman (2014) invented two new file formats for efficient read and write operations while particularly avoiding the HDF dependency. Both proposals only gained experimental acceptance within the MA. As there also was no adoption in the Modelica tool environment, the workaround is to store the simulation result data as CSV file, which then can be read again by the one-dimensional look-up table blocks of `ModelicaTableAdditions`.

2 ExternData

The Modelica library `ExternData` developed out of the need to offer an open-source utility package for efficient parameterization of property parameters from external data resources. It has been successfully tested in Dymola, OpenModelica[13] and SimulationX.

2.1 Library Design

The library (as shown in Figure 1) consists of top-level data source records for each supported file type, the provided accessor functions in `ExternData.Functions` and the external objects in `ExternData.Types`.

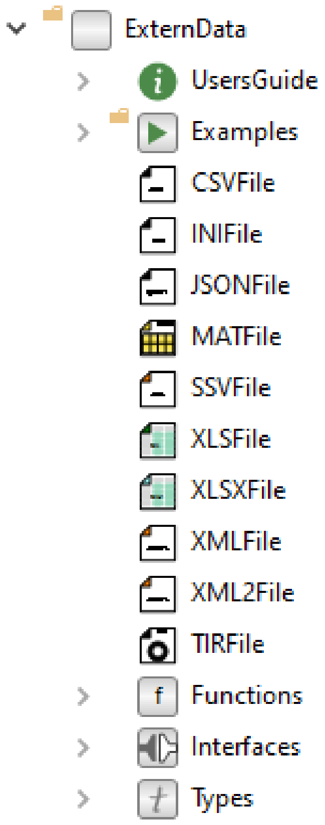


Figure 1. Library structure of `ExternData`

2.1.1 Data Source Records

The data source records are convenience types to encapsulate the external object component with its accessor functions.

A naïve record type definition together with an exemplary user call (application model) is given in Listing 1. (For the sake of clarity, the declaration of the external type `Types.ExtObj` and the external function annotation for `ExtFun` are skipped.)

[9] ExternData Git repository, https://github.com/modelica-3rdparty/ExternData

[10] ModelicaTableAdditions Git repository, https://github.com/modelica-3rdparty/ModelicaTableAdditions

[11] There is no specific name or file extension for the Dymola-specific text/script files starting with "#1" as first line.

[12] EnergyPlus Weather Data, https://energyplus.net/weather

[13] Open Source Modelica Consortium (OSMC), https://openmodelica.org/

Listing 1. Naïve record type definition

```
// Library
record DataSource "Data source record"
  parameter String fileName = ""
    "External data resource";
  final parameter Types.ExtObj obj =
    Types.ExtObj(fileName) "Ext. object";
  pure function get "Accessor function"
    input Types.ExtObj obj "Ext. object";
    input String s "Accessor id";
    output Real out "Data value";
    external "C" out = ExtFun(obj, s);
  end get;
end DataSource;

// Application model
parameter DataSource dataSource(
  fileName="dataSource.ext");
parameter Real p = dataSource.get(
  dataSource.obj, "id");
```

The disadvantage of such a naïve approach is that the handle of the external object, i.e. `dataSource.obj`, has to be passed by every call of the accessor functions despite it actually is an implementation detail of the record and could be a protected component[14].

Listing 2. Sophisticated record type definition

```
// Library
package Functions "Functions"
  pure function get "Accessor function"
    extends Interfaces.getBase;
    external "C" out = ExtFun(obj, s);
  end get;
end Functions;

package Interfaces "Interfaces"
  partial record DataSourceBase
    "Base data source record"
    replaceable function get = getBase;
  end DataSourceBase;
  partial function getBase "Base function"
    input Types.ExtObj obj "Ext. object";
    input String s "Accessor id";
    output Real out "Data value";
  end getBase;
end Interfaces;

record DataSource "Data source record"
  parameter String fileName = ""
    "External data resource";
  final parameter Types.ExtObj obj =
    Types.ExtObj(fileName) "Ext. object";
  extends Interfaces.DataSourceBase(
    redeclare final function get =
      Functions.get(obj=obj)
      "Accessor function");
end DataSource;

// Application model
parameter DataSource dataSource(
  fileName="dataSource.ext");
parameter Real p = dataSource.get("id");
```

A more sophisticated library design is based on clean interfaces for the records and accessor functions enabling inheritance, and thus the possibility of function redeclaration (Beutlich 2018). The general concept is presented in Listing 2.

With such a sophisticated library design the actual implementation (as an external object) is disguised from the caller as the handle of the external object no longer needs to be passed by the member accessor functions.

As of MLS 3.5 (Modelica Language Specification)[15], it is not yet fully specified, if external objects may be used in records[16].

2.1.2 External Functions

The actual external functions and objects serving the Modelica external function interface are implemented in C, i.e. no C++ is utilized.

Independent of the actual file type, the accessor functions for scalars of type `Real`, `Integer`, `Boolean` or `String` are named `getReal`, `getInteger`, `getBoolean` or `getString`, respectively. For one/two-dimensional arrays, the accessor functions are appended by `Array1D`/`Array2D`, e.g., `getRealArray1D` or `getIntegerArray2D`. There also are the corresponding functions to retrieve the array dimensions from the external data resource, i.e., `getArraySize1D` and `getArraySize2D` (and also `getArrayRows2D` and `getArrayColumns2D`).

2.1.3 Structural Parameters

Reading structural parameters from external data resources (as shown for an XML file by Listing 3) by functions `getArraySize1D` or `getArraySize2D` is not generally supported[17]. Of the tested Modelica tools, it only works in SimulationX.

Listing 3. Accessing structural parameters in SimulationX

```
// SimulationX application model
parameter String s = "vector"
  "XML element name";
parameter ExternData.XMLFile dataSource(
  fileName="dataSource.xml")
  "Data source record";
parameter Integer m =
  dataSource.getArraySize1D(s)
  "Structural parameter";
parameter Real p[:] =
  dataSource.getRealArray1D(s, m)
  "Array parameter";
```

To assist the Dymola users, alternative implementations using `readArraySize1D`/`readArraySize2D` functions are available. This comes with the disadvantage of redundant file I/O and is demonstrated by Listing 4.

[14] Only public sections are allowed for records, though.

[15] MA project "Libraries", https://specification.modelica.org/

[16] MLS issue #2399, https://github.com/modelica/ModelicaSpecification/issues/2399

[17] MLS issue #2425, https://github.com/modelica/ModelicaSpecification/issues/2425

Listing 4. Accessing structural parameters in Dymola

```
// Dymola application model
parameter String s = "vector"
  "XML element name";
parameter ExternData.XMLFile dataSource(
  fileName="dataSource.xml")
  "Data source record";
parameter Integer m =
  ExternData.XMLFile.Functions.
    readArraySize1D(
    varName=s,
    fileName="dataSource.xml")
  "Structural parameter";
parameter Real p[:] =
  dataSource.getRealArray1D(s, m)
  "Array parameter";
```

2.1.4 Missing Data

In some cases it may happen, that data of the external resources is missing, for example, an empty cell of an Excel file. By parameter `detectMissingData`, `ExternData` supports four options how to deal with missing data values.

- Return data-type specific defaults
- Return data-type specific defaults and print a message
- Return data-type specific defaults and raise a warning
- Stop the simulation with an error message

Similarly, the accessor functions (Section 2.1.2) also return a Boolean output `exist` to indicate if the retrieved data is available or missing. This way, the reaction on missing data can be modeled per function call.

2.2 Supported File Types

`ExternData` supports various file types for different kind of requirements.

2.2.1 CSV

CSV files contain exactly one data set that can be considered as matrix. An example file with three columns and a header line is given by Listing 5. Both the number of header lines to be ignored and the column delimiter character can be specified.

Listing 5. Example CSV file

```
x,y,z
0,0,0
0.5,0.25,0.125
1,2,3
```

2.2.2 INI

INI files contain scalar properties as key-value-pairs which are grouped by sections. The INI-keys can be fully qualified Modelica names using the dot notation. An example file with the default section and a named section is given by Listing 6.

Listing 6. Example INI file

```
# Default section
gain.k = 1
[Data set]
gain.k = 2
```

2.2.3 JSON

JSON files can be used to define scalars, vectors or matrices which can be arbitrarily structured. The JSON-keys must not contain the dot character to properly work with the accessor functions of `ExternData`. An example file with three different values is given by Listing 7.

Listing 7. Example JSON file

```
{
  "Data set": {
    "gain": {
      "k": "2"
    }
  },
  "vector": [1,2,3],
  "matrix": [[0,0],[0.5,0.25],[1,2]]
}
```

2.2.4 MATLAB MAT (including HDF)

MATLAB MAT files are binary files that can be used for scalars, vectors or matrices. Though it is a proprietary file format it is a common data exchange format for various scientific applications. MATLAB MAT of version 7.3 are HDF5 files and can be considered as a dedicated HDF5 data container. This format version is especially recommended for huge data volumes. However, it is not advisable to read huge arrays as Modelica variables. `ExternData` supports the access of nested structures using dot notation.

2.2.5 SSV

SSV files are standardized XML files that are used within the context of SSP to connect and parameterize FMUs. Certainly, they can not only be used to parameterize imported FMUs in the Modelica simulation environment. One SSV file describes exactly one parameter set where (as of version 1.0) only scalar parameter values are supported. Listing 8 displays an example SSV file.

Listing 8. Example SSV file

```
<?xml version="1.0" encoding="UTF-8"?>
<ssv:ParameterSet version="1.0" name="Data
  set" xmlns:ssv="http://ssp-standard.org/
  SSP1/SystemStructureParameterValues">
  <ssv:Parameters>
    <ssv:Parameter name="gain.k">
      <ssv:Real value="2"/>
    </ssv:Parameter>
  </ssv:Parameters>
</ssv:ParameterSet>
```

Technically, the external object `ExternData.Types.ExternXML2File` is reused while the SSV accessor functions call the appropriate

XML2 functions (`ExternData.Functions.XML2.*`) with dedicated XPath query expressions.

2.2.6 TIR

TIR files define domain-specific tire properties. They are similar to INI files and are implemented to share the same external object `ExternData.Types.ExternINIFile` with respective format permissions.

2.2.7 Excel XLS/XLSX

Both legacy XLS and Office Open XML based XLSX Excel files are supported for parameterization of scalars or matrices.

2.2.8 XML

There are two implementations available.

1. `ExternData.XMLFile` is a straightforward implementation to return values from XML element nodes

2. `ExternData.XML2File` enables full support of XPath query expressions to also query XML attributes or more complicated XML structures.

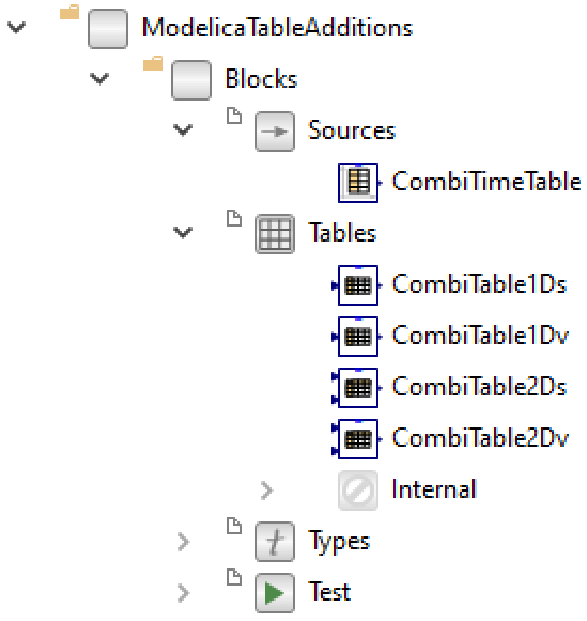
There is no restriction on the underlying XML schema, i.e. it can be used for arbitrarily structured XML data, being standardized (e.g., SSV or CPACS (Common Parametric Aircraft Configuration Schema)[18]) or customized.

3 ModelicaTableAdditions

The Modelica library `ModelicaTableAdditions` developed out of the need to offer reading of external data sources for stimulation parameters (e.g., look-up tables) from commonly used text file formats. Therefore, the blocks of `ModelicaTableAdditions` extend the Modelica Standard Tables of the MSL by support for additional file formats. It does not add other features like N-dimensional arrays, scattered data or spline approximation (as for example by Ungethüm and Hülsebusch (2009)). The Dymola MOS file format, which is the only text file format supported by the Modelica Standard Tables is not suitable for exchange between different applications. This is where CSV and JSON have their advantages, which can easily be processed by different applications. As with the Modelica Standard Tables, the external functions are implemented in pure C (i.e., no C++). For JSON, the same library dependencies are utilized as with `ExternData`. It is possible to use components of packages `ExternData`, `ModelicaTableAdditions` and MSL in the same application model. The library has been successfully tested in Dymola and OpenModelica (Linux only).

3.1 Library Design

The library (as shown in Figure 2) consists of the five look-up table blocks known from the MSL, but this time within the `ModelicaTableAdditions` name-space.



- ModelicaTableAdditions
  - Blocks
    - Sources
      - CombiTimeTable
    - Tables
      - CombiTable1Ds
      - CombiTable1Dv
      - CombiTable2Ds
      - CombiTable2Dv
    - Internal
  - Types
  - Test

Figure 2. Library structure of `ModelicaTableAdditions`

3.2 Supported File Types

3.2.1 CSV

The table of a CSV file (as shown by Listing 5) can be used for time-driven simulation or one/two-dimensional look-up tables. Both the number of header lines to be ignored and the column delimiter character can be specified. Addressing certain columns by their names in the (optional) CSV header line is not supported.

Support for CSV files was also ported to the Modelica Standard Tables and merged to the MSL master branch in early 2021[19].

3.2.2 EPW

The weather conditions of one year are the typical stimulation parameters for building energy simulations, such as the Modelica `Buildings` Library (Wetter et al. 2014). EnergyPlus provides weather data for simulation in various formats, especially their EPW format. Since this file format is not natively supported by the Modelica Standard Tables it manually needed to be pre-processed and converted to either Dymola MOS or MATLAB MAT format.

This pre-processing no longer is necessary as the EPW format is directly supported by all one-dimensional look-up table blocks of the `ModelicaTableAdditions` library.

3.2.3 JSON

Similar as with CSV, tables can be read from JSON files and be utilized as stimulation parameters, e.g. parameter “matrix” of Listing 7.

[18] CPACS, `https://cpacs.de`

[19] MSL issue #3691, `https://github.com/modelica/ModelicaStandardLibrary/pull/3691`

4 Conclusions and Outlook

The open-source libraries `ExternData` and `ModelicaTableAdditions` are an offer to Modelica library developers and users to efficiently parameterize Modelica application models. Its right to exist is due to a missing layered (MA) standard for the parameterization of Modelica models. As already mentioned by Tiller (2005), the benefits of such a standardization are the cutback of library code towards the Modelica tools to even further increase the efficiency, convenience and usability of the parameterization of application models.

There is no support for units so far, i.e., unit conversion is left to the application. This could be also addressed by a standardized parameterization.

Library-wise, one future idea is the support of (symmetrical or asymmetrical) encrypted external resources, which is not yet covered by the MLS. In such cases the external functions require the appropriate (private) key to decrypt the external resources at simulation run-time in memory. Again, encryption as an isolated application can only be considered a short-term solution towards a future standard.

Furthermore, it might be desirable if the mentioned open issues on the MLS regarding external objects could be clarified finally.

Acknowledgements

The authors would like to thank everybody who has contributed to the libraries `ExternData` and `ModelicaTableAdditions`, particularly, Adrian Pop, Hang Yu, Martin Sjölund, Mike Dempsey and Peter Harman.

References

Beutlich, Thomas (2018-08). *Modeling Hints for Modelica External Interfaces*. Presentation given at the 19th Modelisax Meeting, Dresden, Germany. URL: https://tinyurl.com/Modelisax2018Hints.

Beutlich, Thomas, Gerd Kurzbach, and Uwe Schnabel (2014-03). "Remarks on the Implementation of the Modelica Standard Tables". In: *Proceedings of the 10th International Modelica Conference*. Ed. by Hubertus Tummescheit and Karl-Erik Årzén. Lund, Sweden, pp. 893–897. DOI: 10.3384/ecp14096893.

Kellner, Matthias et al. (2006-09). "Parametrization of Modelica Models on PC and Real time platforms". In: *Proceedings of the 5th International Modelica Conference*. Ed. by Christian Kral and Anton Haumer. Vienna, Austria, pp. 267–273. URL: https://www.modelica.org/events/modelica2006/Proceedings/sessions/Session3b2.pdf.

Köhler, Jochen and Alexander Banerjee (2005-03). "Usage of Modelica for transmission simulation in ZF". In: *Proceedings of the 4th International Modelica Conference*. Ed. by Gerhard Schmitz. Hamburg, Germany, pp. 587–592. URL: https://modelica.org/events/Conference2005/online_proceedings/Session7/Session7c1.pdf.

Pfeiffer, Andreas, Ingrid Bausch-Gall, and Martin Otter (2012-09). "Proposal for a Standard Time Series File Format in HDF5". In: *Proceedings of the 9th International Modelica Conference*. Ed. by Martin Otter and Dirk Zimmer. Munich, Germany, pp. 495–506. DOI: 10.3384/ecp12076495.

Reisenbichler, Ulf et al. (2006-09). "If we only had used XML..." In: *Proceedings of the 5th International Modelica Conference*. Ed. by Christian Kral and Anton Haumer. Vienna, Austria, pp. 707–716. URL: https://www.modelica.org/events/modelica2006/Proceedings/sessions/Session6d1.pdf.

Tiller, Michael (2005-03). "Implementation of a Generic Data Retrieval API for Modelica". In: *Proceedings of the 4th International Modelica Conference*. Ed. by Gerhard Schmitz. Hamburg, Germany, pp. 593–602. URL: https://modelica.org/events/Conference2005/online_proceedings/Session7/Session7c2.pdf.

Tiller, Michael and Peter Harman (2014-03). "recon – Web and network friendly simulation data formats". In: *Proceedings of the 10th International Modelica Conference*. Ed. by Hubertus Tummescheit and Karl-Erik Årzén. Lund, Sweden, pp. 1081–1093. DOI: 10.3384/ecp140961081.

Tiller, Michael and Dietmar Winkler (2014-03). "impact - A Modelica Package Manager". In: *Proceedings of the 10th International Modelica Conference*. Ed. by Hubertus Tummescheit and Karl-Erik Årzén. Lund, Sweden, pp. 543–548. DOI: 10.3384/ecp14096543.

Tiller, Michael and Dietmar Winkler (2015-09). "Where impact got going". In: *Proceedings of the 11th International Modelica Conference*. Ed. by Peter Fritzson and Hilding Elmqvist. Versailles, France, pp. 725–736. DOI: 10.3384/ecp15118725.

Ungethüm, Jörg and Dirk Hülsebusch (2009-09). "Implementation of a Modelica Library for Smooth Spline Approximation". In: *Proceedings of the 7th International Modelica Conference*. Ed. by Francesco Casella. Como, Italy, pp. 669–675. DOI: 10.3384/ecp09430013.

Wetter, Michael et al. (2014-07). "Modelica Buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506.

Power Flow Record Structures to Initialize OpenIPSL Phasor Time-Domain Simulations with Python

Sergio A. Dorado-Rojas[1] Giuseppe Laera[1] Marcelo de Castro Fernandes[1] Tetiana Bogodorova[1] Luigi Vanfretti[1]

[1]Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, United States,
{dorads, laerag, decasm3, bogodt2, vanfrl}@rpi.edu

Abstract

This paper presents a tool to populate power flow results for phasor time-domain simulations with the Open Instance Power System Library (OpenIPSL). Our proposal takes advantage of the object-oriented philosophy of Modelica and introduces a data structure based on records to handle power flow data for a given network model. Such records constitute a user-friendly interface to change the guess values used to solve the initial condition of a dynamical simulation straightforwardly. Power flow calculations are carried out by the open-source Python library GridCal. We demonstrate the tool capabilities by generating power flow results for several grid models and comparing them with those obtained via proprietary tools such as PSS/E. Moreover, we provide tutorial materials to ease integrating the tool for a new/experienced OpenIPSL user.

Keywords: GridCal, OpenIPSL, Power Flow, Python, Records

1 Introduction

The Open Instance Power System Library (OpenIPSL) is an open-source library of power system component models written entirely in Modelica (Baudette et al. 2018). Beyond the inherent advantages of the Modelica language, OpenIPSL components are constantly cross-validated against commercial packages such as PSS/E, producing practically the same results (Laera 2016) and exhibiting the same or even better simulation performance (see Henningsson, Olsson, and Vanfretti (2019) and Dorado-Rojas, Navarro Catalán, et al. (2020)).

Successful use cases of the library come from a broad range of applications such as multi-domain simulation (Gomez et al. 2018), damping (Boersma et al. 2020) and parameter estimation (Podlaski et al. 2020) in power systems, dynamic stability assessment (Nohac et al. 2019), co-simulation for energy analysis (Gusain, Cvetkovic, and Palensky 2019), stability analysis of hydro-power grids (Winkler 2019), wind turbine control (Qin et al. 2019), cyber-attack evaluation (Pan, Gusain, and Palensky 2019), power system stability enhancement (Gonzalez-Torres et al. 2019), extremum seeking control (Müller et al. 2020), and data generation for machine learning applications (Dorado-Rojas, de Castro Fernandes, and Vanfretti 2020).

Motivation

Despite the library's usefulness, the main caveat is the absence of a systematic approach to link phasor time-domain simulations with static computations like power flows. Power flow computations are ubiquitous in any power system analysis. A power flow problem involves determining the system's voltage profiles and electrical power transfer across a network given the generator power injections and load consumption. Mathematically, it is a nonlinear vector algebraic equation commonly solved using an iterative method such as a Newton-Raphson algorithm. From the dynamical perspective, a power flow result represents an operating condition for which may contain a *potential* equilibrium for the underlying dynamical system. So, the power flow result represents the set of initial guesses to initialize a dynamic model and analyze an electrical grid's behavior subjected to a dynamical event. Observe that because the simplified algebraic representation of the power grid in the power flow problem, many of its solutions can result in operating conditions where an equilibrium may not exist when the system's dynamical model is considered.

OpenIPSL models contain a myriad of nonlinearities employed to represent dynamical behaviors more accurately. So, varying the initial condition of a dynamical simulation represents a critical step towards system assessment. So far, users have proposed ad-hoc solutions to generate power flow results (e.g., using Matpower[1] or PSS/E[2] as in Vanfretti et al. (2017)). However, despite valuable, these efforts do not completely fill the gap to easily provide power flow solutions to OpenIPSL models. The former approach replaces the power flow values in the `*.mo` file of the model directly, which is inconvenient from the user's point of view. The latter depends on proprietary software which might not be available to the base users of OpenIPSL. The OpenIPSL community, and users of other Modelica-based power system libraries (see Winkler (2017)), will more than welcome a systematic power flow approach based on open-source tools to integrate into their models quickly. Addressing this issue is the primary purpose of this paper.

[1] See https://github.com/dgusain1/InitialiseModelica

[2] See https://github.com/ALSETLab/Raw2Record

DOI
10.3384/ecp21181147

Contribution

This paper's main contribution is to bridge the gap between phasor time-domain simulation and static computations for OpenIPSL utilizing an open-source-based pipeline.

We propose a Modelica records structure to handle all power flow variables. Such nested records data structure enables a user to replace a power flow condition, typically composed of several algebraic variables, with a single click or one line of code. These records are created automatically from the model's `*.mo` file.

GridCal[3], and provides open-source Python library for power system computations, such as solving the power flow problem. A remarkable characteristic of GridCal is its built-in PSS/E parser. Consequently, users can parse `*.raw` files containing a grid static model's information to a GridCal internal grid representation. In our examples, we will use PSS/E files to construct GridCal models. This enables us to benchmark the GridCal power flow results against PSS/E outputs. By doing so, we bring confidence to Modelica tools in terms of the quality of results, showing that Modelica-based power system models can be initialized and result in the same initial condition and provide the same simulation results as proprietary tools.

So far, we summarize the main contributions of our work as follows:

- we bridge the gap between Modelica-based tools and conventional domain-specific power system tools;
- we automate the process of providing good initial guess values to solve the initialization process of Modelica-based power system models with the OpenIPSL library;
- we make Modelica-based tools more attractive for dynamic power system simulation, making it easier for users to use OpenIPSL models for analysis under multiple operating conditions by a power flow record structure;
- with the contributions above, we facilitate the potential adoption of and transition to Modelica-based tools by power system domain specialists.

Paper Structure

This paper is structured as follows: Section 2 gives a brief introduction to the power flow problem in electrical networks. In Section 3, we introduce the records structure proposed to handle power flow variables. We illustrate how this data container can be linked to OpenIPSL models in Section 4, where we also benchmark the power flow values against the results obtained with commercial tools. Finally, Section 5 concludes the work.

2 The Power Flow Problem

The *power flow* (also incorrectly called *load flow*) problem is undoubtedly one of the most performed calculations in power system applications (Stott 1974). For instance, these calculations are carried out many times in Operation and Planning procedures for power grids. An application of a particular interest to this paper is that power flow solutions provide a *potential* starting guess to solve the initialization problem at which a dynamic simulation may start. Hence, a power flow can be considered one of the most critical problems to be solved when studying a power system (Milano 2009).

The problem, however, is not new, and nor are the techniques used to solve it. The first practical solutions began to appear in the mid-1950s with the aid of digital computers (Ward and Hale 1956) and a breakthrough came about a decade later. The development of incredibly efficient handling of sparse matrices (Tinney and Walker 1967) was paramount to the wide adoption of Newton-Raphson (NR) algorithm. As new issues to solve the power flow problem have arisen, a myriad of new techniques and methods have been proposed; however, NR-based techniques are still the most preeminent methods (Stott 1974; Milano 2009).

From the mathematical perspective, a power flow problem is posed as a set of nonlinear algebraic equations. Its solution will determine an operational point for a specified loading and generation condition in the power system. This operational point is defined by the voltage magnitude and angle in each bus of the system, together with the active and reactive powers generated and consumed in generation and load buses respectively. In the current paper, we will give a brief introduction to its formulation.

In most power flow formulations, the power system is assued to be perfectly balanced and operating at constant frequency (i.e. 50/60 Hz) which would allow it to be represented in its positive sequence equivalent circuit (Stott 1974). If a system can be represented in its positive sequence equivalent, it is then possible to assemble its nodal admittance matrix $\mathbf{Y}$ and to write the *nodal equation* as follows:

$$\bar{\mathbf{I}} = \mathbf{Y}\bar{\mathbf{V}}, \tag{1}$$

where $\bar{\mathbf{I}}$ is the nodal injection current phasor vector, $\bar{\mathbf{V}}$ is the nodal voltage phasor vector and $\mathbf{Y}$ is the admittance matrix, which is square and sparse.

We could use the nodal equation to compute the voltage at all nodes if all current injection measurements were available. Unfortunately, this is not the case in an electric grid where the known quantities differ from bus to bus. For instance, in a load node, active and reactive power consumption (P,Q) are assumed to be known. Likewise, in a generation bus, active power injection and voltage magnitude at the generator terminals are typically known (P,V). Then, the nodal equation has to be reformulated in terms of P,Q, and V. Because the steady-state relationship between power and voltage/current is nonlinear (and

[3] GridCal is able to import and parse model description and parameter files from proprietary software such as PSS/E and DigSilent, and also widespread open-source libraries for power system analysis like Matpower. In contrast to many proprietary electrical grid software tools, GridCal runs on Windows, Linux, and macOS natively. It can be downloaded from https://github.com/SanPen/GridCal

complex-valued), the linear nodal equation into a nonlinear set of complex-valued equations on P, Q, V, and the voltage phasor angle θ.

The exact formulation in the power system jargon is the following. For the mth bus, four variables are either specified or should be calculated in a power flow: active power injected in the bus in per unit P_m, reactive power injected in the bus in per unit Q_m, node voltage phasor magnitude in per unit V_m and node voltage phasor angle in radians θ_m. In load buses (identified as PQ buses) the variable that is known beforehand is the specified apparent power ($S_m^{\text{sp}} = P_m^{\text{sp}} + jQ_m^{\text{sp}}$), while in generation buses (called PV buses) the specified variables are the voltage magnitude ($\bar{V}_m$) and active power (P_m^{sp}). In addition to that, there should be one *slack bus* which should have a specified voltage magnitude value and, most importantly, it should be responsible for providing the angle reference used for all other calculations (Stott 1974).

The power flow solution is achieved when the computation of active and reactive power via the nodal equation, using the solution values from the most recent iteration, matches the given data for active and reactive power. In other words, the solution occurs when the mismatch between specified and calculated power values is less than or equal to some given tolerance. We can write such a mismatch as

$$\Delta S_m = S_m^{\text{sp}} - V_m I_m^* = P_m^{\text{sp}} + jQ_m^{\text{sp}} - \bar{V}_m \sum_{k \in \mathbf{K}_m} Y_{mk}^* \bar{V}_k^*, \quad (2)$$

where $\mathbf{K}_m$ is the set of buses k which are directly connected to bus m and the superscript $(^*)$ denotes the complex conjugate. By using the fact that $Y_{mk} = G_{mk} + jB_{mk}$ and expressing the phasor $\bar{V}_m$ as $\bar{V}_m = V_m(\cos\theta_m + j\sin\theta_m)$, we can split Equation (2) into its real and imaginary parts as:

$$\begin{cases} \Delta P_m = P_m^{\text{sp}} - V_m \sum_{k \in \mathbf{K}_m} (G_{mk}\cos\theta_{mk} + B_{mk}\sin\theta_{mk}) V_k, \\ \Delta Q_m = Q_m^{\text{sp}} - V_m \sum_{k \in \mathbf{K}_m} (G_{mk}\sin\theta_{mk} - B_{mk}\cos\theta_{mk}) V_k, \end{cases}$$

where $\theta_{mk} = \theta_m - \theta_k$. Note that, in this polar formulation, the unknown variables are the nodal voltage phasor magnitudes (V_m) and angles (θ_m).

As said previously, a power flow problem is typically solved using an NR algorithm. First, a nonlinear vector function $\mathbf{f} : \mathbb{R}^{2n} \mapsto \mathbb{R}^{2n}$ is defined, where n is the total number of nodes (buses). This function could be expressed as:

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} \Delta\mathbf{P} \\ \Delta\mathbf{Q} \end{bmatrix}, \quad (3)$$

where the $\Delta\mathbf{P}$ and $\Delta\mathbf{Q}$ are the n-row vectors $[\Delta P_m]$ and $[\Delta Q_m]$, respectively. In addition, $\mathbf{x} \in \mathbb{R}^{2n}$ is given by

$$\mathbf{x} = \begin{bmatrix} V_1 & \cdots & V_m & \cdots & V_n & \theta_1 & \cdots & \theta_m & \cdots & \theta_n \end{bmatrix}^T, \quad (4)$$

where the superscript ^{T} stands for transpose. To find the power flow solution, we state the following equation (Milano 2009).

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \quad (5)$$

Due to the nonlinear nature of $\mathbf{f}$, it is impossible to find a closed-form solution for such a problem. Therefore, it is necessary to use an iterative method such as a classical NR algorithm. The i-th iteration of the NR method is written as (Milano 2009)

$$\begin{cases} \Delta\mathbf{x}_i = [\mathbf{J}(\mathbf{x}_i)]^{-1}\mathbf{f}(\mathbf{x}_i), \\ \mathbf{x}_{i+1} = \mathbf{x}_i + \Delta\mathbf{x}_i, \end{cases} \quad (6)$$

where $\mathbf{J}(\mathbf{x})$ is the Jacobian matrix of function $\mathbf{f}$. The iterative method will stop when $\mathbf{f}(\mathbf{x})$ is sufficiently close to $\mathbf{0}$ or, in other words, when its norm is less than a tolerance set by the user. Besides, there are many different ways to find $\mathbf{x}_0$, which is used to start the process described in (6). Generally, robust techniques usually allow to find a solution when using a flat start, i.e., when all voltage magnitudes are started as 1 per unit and all angles are started as 0 radians, or one could use a previous power flow solution can be used for the same system.

3 Power Flow Records Structure

One of the Modelica language's main advantages is the object-oriented paradigm that enables the user to create dynamic system models hierarchically. Such a structure allows the user to manage model parameters systematically.

A Modelica `record` is a data container used to store a wide range of information about a model, such as parameter values, simulation settings, or values of specific variables for several analysis conditions. Records permit changing a significant number of variables of a given model by modifying just one parameter that related hierarchically to many variables inside the data representation.

Records are a perfect structure for handling power flow values in a dynamic simulation. Suppose power flow results are handled using a record-based data structure. In that case, we could modify the power flow condition of an OpenIPSL model by varying only one attribute of the model, namely, the power flow record, rather than individually changing multiple variables.

The proposed power flow record structure is presented in Figure 1. A Python script called `create_pf_records` creates the Modelica files containing the `record` structure and places them inside the model's root folder in a directory called `PF_Data`. This function reads the `.mo` file of the model (`<model_name>.mo`) as a plain text file and uses several regular expressions to determine the number of buses, generators, loads, and transformers in the network. Such a script becomes handy when a user has an existing OpenIPSL model and would like to add a power flow records structure automatically.

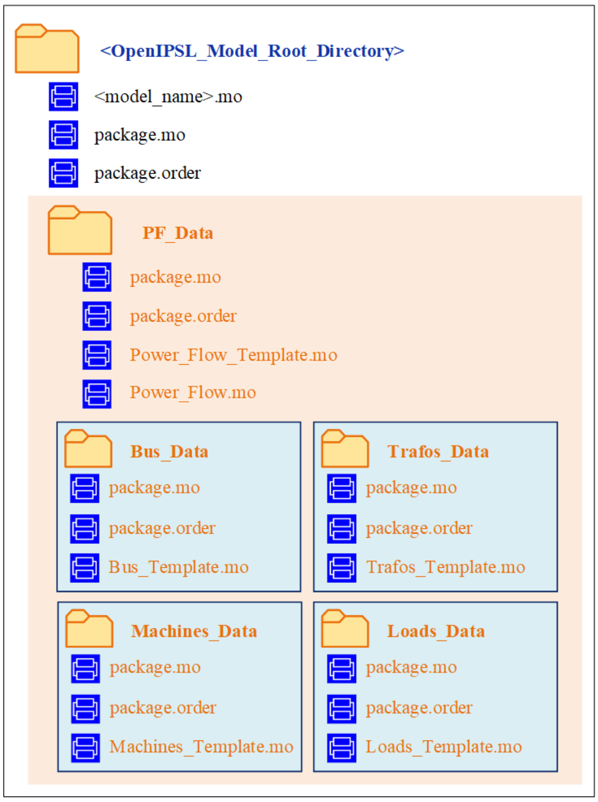


Figure 1. File structure of the power flow records inside the OpenIPSL model directory.

We define our power flow record structure in the class **Power_Flow** shown in Figure 2. **Power_Flow** is a record having a single attribute: a **replaceable record PowerFlow**. A replaceable condition allows the user having many power flow results for the same model (see Figure 4).

The main idea behind the proposed nested structure is that, by setting the value of **PowerFlow**, the user changes all the power flow variables at once. So, a model has a unique **Power_Flow** record whose power flow attribute is replaceable.

PowerFlow has four attributes, which are also records themselves: a record for bus voltages and angles (**Bus_Data**), another for transformer tap positions (**Trafos_Data**), a third one for active and reactive power consumption (**Load_Data**), and a fourth record for machine power dispatch (**Machines_Data**). Naturally, the number of variables inside each of these internal records depends on each particular power system model. For each record type, the variables are specified by the **partial record** templates called **Bus_Template**, **Trafos_Template**, **Machines_Template**, and **Loads_Template**, respectively.

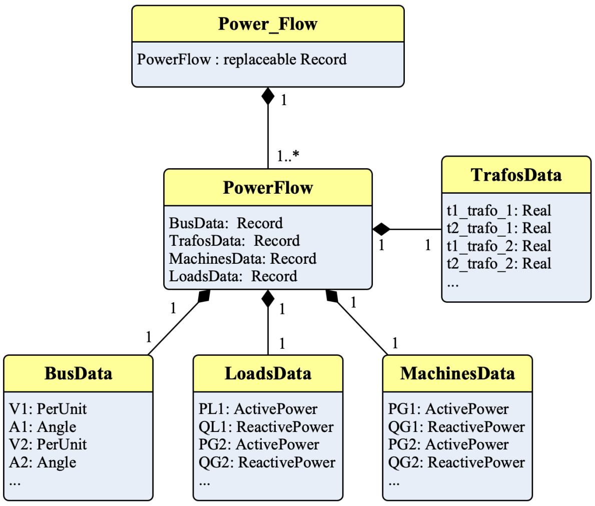


Figure 2. Class diagram for the proposed power flow record structure.

The numerical results are written by a parser function that translates the power flow result from a GridCal model computation into a format compatible with the Modelica record structure. This function is called **gridcal2rec**. **gridcal2rec** creates a **PowerFlow** instance placed inside `PF_Data`, whose attributes are four record instances: **Bus_Data**, **Trafos_Data**, **Machines_Data**, and **Loads_Data**.

4 Computing and Linking PF Records

This section describes how the records structure, illustrated previously, can be successfully applied to grid models of different sizes.

4.1 Creation of Records Structure

A user can integrate our proposed power flow structure into any existing OpenIPSL model using the code contained within the `pf2rec` library (available on GitHub). The records structure is instantiated by the `create_pf_records` function. Listing 1 presents a minimal example of creating a power flow record for the Single Machine Infinite Bus (SMIB) system.

Listing 1. Creation of the records structure

```
from pf2rec import *
import os

# Current working directory
_wd = os.getcwd()
_model_package = 'SMIB'

# Path to the model package directory
data_path = os.path.join(_wd, _model_package)
data_path = os.path.abspath(data_path)

path_mo = os.path.join(data_path,
    'SMIB_Base_case.mo')
path_mo = os.path.abspath(path_mo)

# Creating records structure
create_pf_records(_model_package, path_mo,
    data_path,
    openipsl_version = '2.0.0')
```

Note that the content of the Modelica model is saved within several `*.mo` files. This practice is encouraged since it allows to increase the complexity of the data layer of the model. As supplementary material to this paper, we provide a tutorial[4] showing the step-by-step construction of the SMIB system, the corresponding power flow records integration, and computation using GridCal.

The four arguments that we pass to **create_pf_records** are the name of the containing package (`_model_package`), the path to the `*.mo` file where the model is declared (`path_mo`), the path to the containing folder of the model package (`data_path`), and the OpenIPSL library version on which the model has been developed (`_version`). Here, the paths are constructed as absolute references thanks to the `os` library. Such a workaround is recommended to avoid any path problems since the records instantiation involves file/folder creation. The script places all the power flow record `*.mo` files inside a new directory called `PF_Data`. `PF_Data` is also added to the `package.order` file of the root package. In this way, the **record** structure is loaded with the model automatically. Once the power flow is created, the data structure in Figure 1 is shown as a nested subpackage, illustrated in Figure 3.

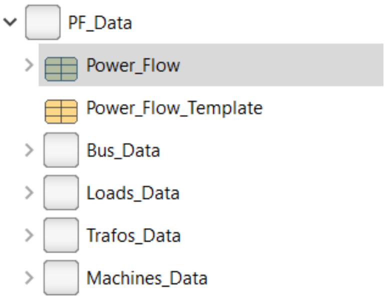


Figure 3. Power flow record structure as a nested subpackage in the model structure.

4.2 Power Flow Computation with GridCal

GridCal is a Python-based object-oriented software for the computation of power flow results. An example of using GridCal to compute power flow and Python to write the power flow solution into record is shown in Listing 2. In this case, a PSS/E `*.raw` file containing the static model information is translated into a GridCal object using the built-in parser class **FileOpen**. The `*.raw` contains the static model of the network, which is required for any power flow formulation. The `*.raw` parser allows us to benchmark the performance of GridCal against PSS/E in terms of power flow result accuracy. Furthermore, this feature reduces the cost of migrating a model from PSS/E to OpenIPSL since the user could initialize both models from the same `*.raw` file. However, the user can define their own grid models from scratch. The reader is referred to the GridCal documentation for network implementation examples.

After creating the grid object via the parser class, an instance of the **PowerFlowDriver** is declared: **pf**. **pf** is responsible for carrying out the power flow computation following user-specified settings (**options**). Recall from Equation (6) that the method for a power flow computation is constrained by the grid topology (i.e., the matrix $\mathbf{J}(\mathbf{x})$). Therefore, the **grid** object must be passed to the **PowerFlowDriver** constructor method for any power flow computation. The PSS/E `*.raw` file can store up to one power flow result. We take advantage of this fact and use that power flow as an initialization value for a base-case power flow computation in GridCal. The result of this base case should be the same power flow (within the solver's tolerance) as the one included in the PSS/E file. The power flow calculation is commanded by invoking the function **pf.run()**. The results are stored as an attribute of the **PowerFlowDriver** class.

Listing 2. Generation of power flow result using GridCal (PSS/E file input)

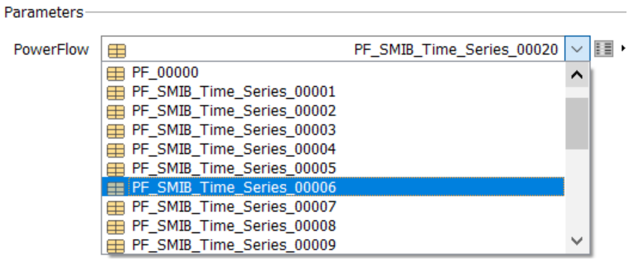
```
_wd = os.getcwd() # working directory
_model_package = 'SMIB'

# Path to the model package directory
data_path = os.path.join(_wd, _model_package)
data_path = os.path.abspath(data_path)

# Path to the PSSE `.raw` file
psse_raw_path = os.path.join(data_path, "
    PSSE_Files", "SMIB_Base_Case.raw")
psse_raw_path = os.path.abspath(psse_raw_path)

# Grid model in GridCal
file_handler = FileOpen(psse_raw_path)

# Creating grid object and setting options
grid = file_handler.open()
options = PowerFlowOptions(SolverType.NR,
    verbose = True,
    initialize_with_existing_solution = False,
    multi_core = False,
    tolerance = 1e-6,
    max_iter = 99,
    control_q = ReactivePowerControlMode.Direct)

pf = PowerFlowDriver(grid, options)
pf.run()

# Writing power flow results in records
gridcal2rec(grid = grid, pf = pf, model_name = '
    SMIB',
    data_path = data_path,
    pf_num = 0,
    export_pf_results = False)
```

Finally, the function **gridcal2rec** takes the grid information and the power flow driver information and writes the results as Modelica records, following the structure described in Section 3. The new files are placed within the `PF_Data` subfolder, housing the power flow record structure. They are also written automatically in-

[4] https://github.com/ALSETLab/SMIB_Tutorial/ and https://youtu.be/4qfKw9SAXFY

side the corresponding `package.order` file to become available to the user right after the computation is completed. The function **`gridcal2rec`** can be included in automation loops to perform a time series power flow. The resulting output is shown in Figure 4.

In Figure 5 one can notice how the power flow condition, defining several variables in a model, can be set either from the graphical interface or by redeclaring a single parameter in the text layer. To the authors' best knowledge, such a feature is not typically available in commercial power system software for dynamics. However, we can easily incorporate it into OpenIPSL models by exploiting the flexibility of object-oriented structure of the Modelica language.



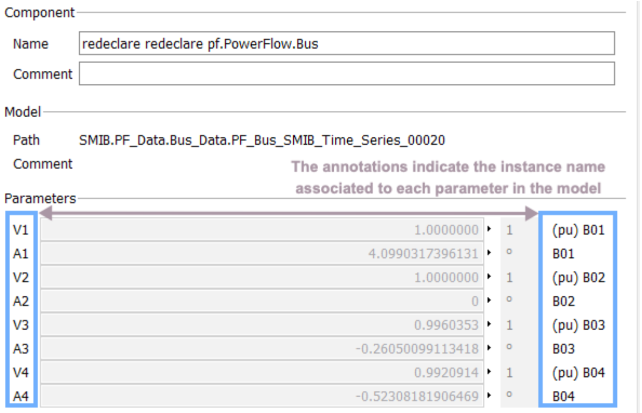
Parameters

PowerFlow: PF_SMIB_Time_Series_00020

- PF_00000
- PF_SMIB_Time_Series_00001
- PF_SMIB_Time_Series_00002
- PF_SMIB_Time_Series_00003
- PF_SMIB_Time_Series_00004
- PF_SMIB_Time_Series_00005
- PF_SMIB_Time_Series_00006
- PF_SMIB_Time_Series_00007
- PF_SMIB_Time_Series_00008
- PF_SMIB_Time_Series_00009

Figure 4. Multiple power flows within an OpenIPSL model.

A possible difficulty of our power flow generation tool is that the user must connect the power flow parameters in each device to the record manually. After several attempts, we noticed that it depended on how the user constructed a particular model, which is unpredictable. However, we included informative annotations in the record attributes to link the initialization values (see Figure 5) correctly.



Component

Name: redeclare redeclare pf.PowerFlow.Bus

Comment:

Model

Path: SMIB.PF_Data.Bus_Data.PF_Bus_SMIB_Time_Series_00020

Comment:

The annotations indicate the instance name associated to each parameter in the model

Parameters

V1	1.0000000	1	(pu) B01
A1	4.0990317396131	°	B01
V2	1.0000000	1	(pu) B02
A2	0	°	B02
V3	0.9960353	1	(pu) B03
A3	-0.26050099113418	°	B03
V4	0.9920914	1	(pu) B04
A4	-0.52308181906469	°	B04

Figure 5. Informative annotations to assist the user link the record attributes to the model correctly.

Despite this caveat, referencing of the power flow variables to the record must be done only *once*. Afterwards, the user must change the Powerflow attribute, not the record itself. Since the references point to the record on the top layer, they remain unchanged. A detailed example of this process in the tutorial[5] accompanying this paper.

4.3 Scalability for Larger Models

We validated our approach in several systems of different number of buses (that defines the scale of the power flow problem) and of state variables (that defines the size of the complexity of dynamic simulation problem). Table 1 and Figure 6 summarize the characteristics of the benchmarked systems and illustrate the tool's performance in terms of execution time for record generation and power flow computation. The results correspond to the best scenario over 100 repetitions with 100 loops each. All models are available within the Application Examples of OpenIPSL.

Table 1. Scalability results on different systems

System (Buses)	*Number of Variables*		*Avg. Execution Time (over 100 loops)*	
	Algebraic	*State*	*Record Creation*	*Power Flow Computation*
SMIB (4)	99	9	4.08 ms ± 255 μs	31.6 ms ± 1 ms
IEEE 9 (9)	241	29	7.29 ms ± 287 μs	35.5 ms ± 1.55 ms
Kundur Two Areas (11)	244	20	5.07 ms ± 194 μs	37.4 ms ± 1.01 ms
AVRI (14)	16	233	5.77 ms ± 144 μs	35.9 ms ± 1.17 ms
Nordic 44 (44)	1294	6315	55.2ms ± 874 μs	349 ms ± 12.8 ms

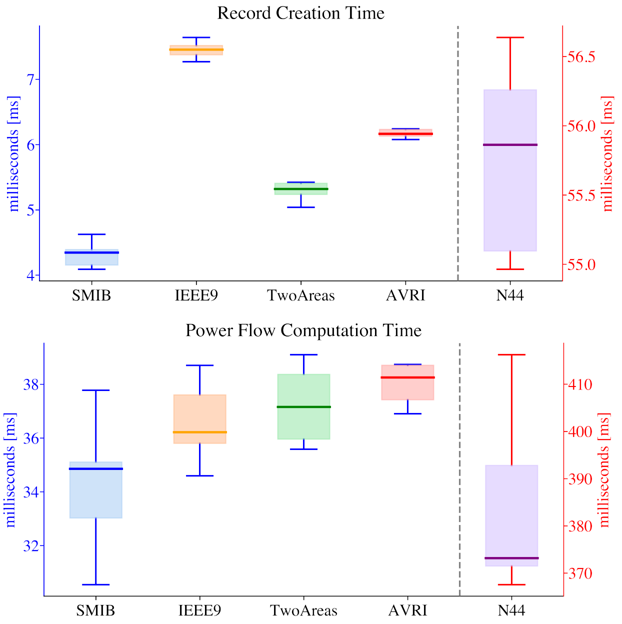


Figure 6. Execution time for record creation (top) and power flow computation (bottom). Observe that the results for the N44 are presented on a different scale.

[5] https://youtu.be/RMD8WEOi6r4

Table 2. Power Flow Comparison between PSS/E and GridCal

System	*Bus*	*Voltage*						*Power*					
		Magnitude [pu]		*Absolute Error*	*Angle [deg]*		*Absolute Error*	*P [MW]*		*Absolute Error*	*Q [Mvar]*		*Absolute Error*
		PSS/E	*GridCal*		*PSS/E*	*GridCal*		*PSS/E*	*GridCal*		*PSS/E*	*GridCal*	
SMIB	1	1.0000	1.0000	-9.99×10^{-16}	4.04628	4.04627	-2.24×10^{-6}	40.000	40.000	0.00000	5.417	5.417	3.66×10^{-8}
	2	1.0000	1.0000	0.00000	0.00000	0.00000	0.00000	10.017	10.017	5.63×10^{-6}	8.007	8.007	3.83×10^{-8}
IEEE9	1	1.0400	1.0400	0.00000	0.00000	0.00000	0.00000	71.613	71.613	1.11×10^{-5}	25.592	25.592	4.12×10^{-6}
	2	1.0300	1.0300	0.00000	9.18220	9.18219	-4.36×10^{-6}	163.000	163.000	0.00000	8.925	8.925	-3.69×10^{-6}
	3	1.0250	1.0250	0.00000	4.64766	4.64766	-2.20×10^{-6}	85.000	85.000	0.00000	-12.503	-12.503	-1.23×10^{-5}
Two Areas	1	1.0300	1.0300	0.00000	27.07087	27.07086	-7.19×10^{-6}	700.000	700.000	0.00000	185.035	185.035	-2.56×10^{-5}
	2	1.0100	1.0100	0.00000	17.30648	17.30647	-7.33×10^{-6}	700.000	700.000	0.00000	234.624	234.624	-2.10×10^{-5}
	3	1.0300	1.0300	0.00000	0.00000	0.00000	0.00000	719.095	719.095	-2.58×10^{-5}	176.040	176.040	2.24×10^{-5}
	4	1.0100	1.0100	-9.99×10^{-15}	-10.19216	-10.19215	1.09×10^{-5}	700.000	700.000	0.00000	202.114	202.114	-4.49×10^{-5}
AVRI	1	1.0500	1.0500	0.00000	0.00000	0.00000	0.00000	-100.000	-100.000	0.00000	41.391	41.391	-4.25×10^{-6}
	8	1.0500	1.0500	0.00000	47.01978	47.01976	-1.89×10^{-5}	50.000	50.000	0.00000	19.795	19.795	-7.89×10^{-7}
	12	1.0500	1.0500	0.00000	43.26172	43.26170	-2.11×10^{-5}	50.000	50.000	0.00000	21.916	21.916	-4.04×10^{-6}
N44	3115	1.0000	1.0000	0.00000	-13.59220	-13.59220	1.12×10^{-6}	1114.875	1114.875	0.00000	-395.702	-395.702	1.37×10^{-5}
	6000	1.0050	1.0050	0.00000	-18.37864	-18.37864	-2.86×10^{-6}	1010.808	1010.808	0.00000	-400.800	-400.780	1.97×10^{-2}
	6500	1.0000	1.0000	0.00000	-25.88593	-25.88593	-3.62×10^{-6}	1093.284	1093.284	0.00000	882.375	882.375	-1.36×10^{-4}
	8500	1.0200	1.0200	-9.99×10^{-15}	-5.72443	-5.72443	5.95×10^{-7}	1952.664	1952.664	0.00000	596.683	596.683	2.58×10^{-4}

The Record Creation (RC) process is 5–7x faster than the power flow computation, as expected[6]. Both procedures scale up with the number of algebraic variables, directly related to the dimensionality of the power flow problem. Notice that increase in execution time to generate the records shows an exponential trend with respect to the size of the power flow problem (Figure 7), as expected.

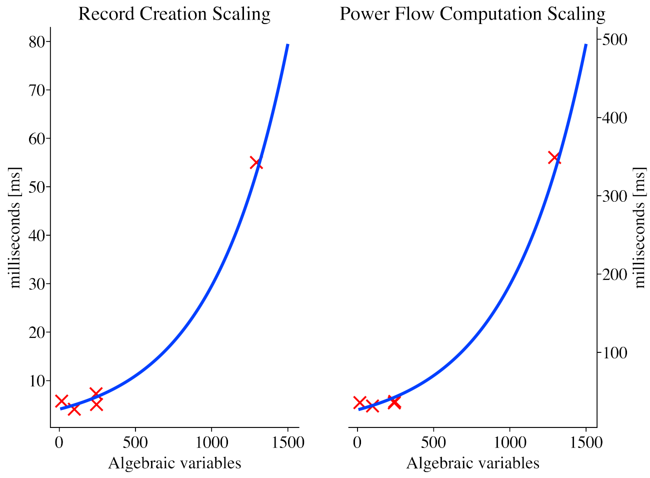


Figure 7. Exponential increase in execution time as a function of the number of algebraic variables in the model.

4.4 Result Validation with PSS/E

The validation against PSS/E of the power flow results obtained using GridCal has been performed on several test systems. In Table 2, a list of the tested networks is given. For each of the networks some buses have been selected indicating their voltage magnitude and angle, the injected/absorbed active and reactive powers of the generating units connected to the corresponding node. Those power flow results are compared with the corresponding calculations obtained from PSS/E including evaluation of an absolute error between the evaluated power flow and reference PSS/E power flow. The power flow values match with low tolerance errors that in some cases hit the machine precision. This shows the validity of the proposed approach of power flow calculation using the open-source Python library GridCal.

5 Conclusions

This article presents an approach to form a record-based data structure to handle power flow starting guesses for a dynamic simulation using the phasor-domain OpenIPSL library. A power flow computation, performed before running a phasor-domain simulation, specifies the starting equilibrium of the nonlinear system simulation. The record class architecture benefits directly from the object-oriented paradigm of the Modelica language, allowing management of *all* power flow variables from a *single* attribute in the model, a feature not common in specialized proprietary power system tools. Such structure can be extrapolated to other open-source Modelica-based power system libraries.

We provide a Python script to create the structure for any existing OpenIPSL model built on versions 1.5.0 or 2.0.0, in this way, naturally expanding capabilities of the library to perform dynamic simulations for different power flow initial conditions. The power flow record instances can be populated by an open-source Python library called GridCal, capable of producing numerically the same results as PSS/E for power flow computations. We also introduce a script to convert the GridCal power flow results to records directly.

From our perspective, the proposed methodology can be useful for users of existing OpenIPSL models, especially for those who study the behavior of the models under different power flow conditions. However, for large scale models the user would have to spend significant time linking the power flow variables to the record. To avoid the aforementioned issue, a model translation tool that translates the information from PSS/E `*.dyr` and

[6] The experiments were performed on an Intel Core i5 Quad-Core (2.0 GHz) processor, with 16 GB RAM DDR4 memory.

`*.raw` files into OpenIPSL `*.mo` models is currently under development. The tool will include the proposed record structure in this paper by default. In that case, the power flow variables will point to the record automatically. This will be a key advantage in helping power system analysts with the potential adoption and transition to Modelica-based tools.

Acknowledgements

The authors would like to thank Fernando Fachini, Ricardo Rincón Ballesteros, and Manuel Navarro Catalán for their support and feedback with the implementation of the software. We would also like to thank Santiago Peñate Vera for developing GridCal.

This research was funded in part by the New York State Energy Research and Development Authority (NYSERDA) through the Electric Power Transmission and Distribution (EPTD) High Performing Grid Program under agreement numbers 137940, 137951, in part by the Engineering Research Center Program of the National Science Foundation and the Department of Energy under Award EEC-1041877, in part by the CURENT Industry Partnership Program, in part by the Center of Excellence for NEOM Research at King Abdullah University of Science and Technology, and in part by Dominion Energy.

Parts of this material are based upon work supported by the U.S. Department of Energys Office of Energy Efficiency and Renewable Energy (EERE) under the Advanced Manufacturing Office, Award Number DE-EE0009139.

Legal Disclaimer

The views expressed herein do not necessarily represent the views of the U.S. Department of Energy or the United States Government, nor of any of the funding bodies listed in the acknowledgement.

References

Baudette, Maxime et al. (2018-01). “OpenIPSL: Open-Instance Power System Library Update 1.5 to iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations”. In: *SoftwareX* 7, pp. 34–36. DOI: 10.1016/j.softx.2018.01.002.

Boersma, S. et al. (2020-09). “Enhanced Power System Damping Estimation via Optimal Probing Signal Design”. In: *2020 22nd European Conference on Power Electronics and Applications (EPE’20 ECCE Europe)*. IEEE, pp. 1–10. DOI: 10.23919/EPE20ECCEEurope43536.2020.9215892.

Dorado-Rojas, Sergio A., Marcelo de Castro Fernandes, and Luigi Vanfretti (2020). “Synthetic Training Data Generation for ML-based Small-Signal Stability Assessment”. In: *2020 IEEE SmartGridComm*.

Dorado-Rojas, Sergio A., Manuel Navarro Catalán, et al. (2020-11). “Performance Benchmark of Modelica Time-Domain Power System Automated Simulations using Python”. In: *Proceedings of the American Modelica Conference 2020*.

Gomez, Francisco J. et al. (2018). “Multi-Domain Semantic Information and Physical Behavior Modeling of Power Systems and Gas Turbines Expanding the Common Information Model”. In: *IEEE Access* 6, pp. 72663–72674. DOI: 10.1109/ACCESS.2018.2882311.

Gonzalez-Torres, J.C. et al. (2019). “Power system stability enhancement via VSC-HVDC control using remote signals: application on the Nordic 44-bus test system”. In: *15th IET International Conference on AC and DC Power Transmission (ACDC 2019)*. Institution of Engineering and Technology, 78 (6 pp.)–78 (6 pp.) ISBN: 978-1-83953-007-4.

Gusain, Digvijay, Milos Cvetkovic, and Peter Palensky (2019-06). “Energy Flexibility Analysis using FMUWorld”. In: *2019 IEEE Milan PowerTech*. IEEE, pp. 1–6. ISBN: 978-1-5386-4722-6. DOI: 10.1109/PTC.2019.8810433.

Henningsson, Erik, Hans Olsson, and Luigi Vanfretti (2019-02). “DAE Solvers for Large-Scale Hybrid Models”. In: pp. 491–502. DOI: 10.3384/ecp19157491.

Laera, Giuseppe (2016). *Modelica Norwegian Grid Models for iTesla and Model Validation Tasks*. Tech. rep.

Milano, Federico (2009). “Continuous Newton’s method for power flow analysis”. In: *IEEE Transactions on Power Systems* 24.1, pp. 50–57.

Müller, Joscha et al. (2020-11). “A Modelica Library for Continuous and Discrete Extremum Seeking for Static and Dynamic Systems”. In: pp. 36–45.

Nohac, Karel et al. (2019-05). “Open Source Platforms for Dynamic Stability Assessment”. In: *2019 20th International Scientific Conference on Electric Power Engineering (EPE)*. IEEE, pp. 1–6. ISBN: 978-1-7281-1334-0.

Pan, Kaikai, Digvijay Gusain, and Peter Palensky (2019-01). “Modelica-Supported Attack Impact Evaluation in Cyber Physical Energy System”. In: *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*. IEEE, pp. 228–233. ISBN: 978-1-5386-8540-2.

Podlaski, Meaghan et al. (2020-11). “Parameter Estimation of User-Defined Control System Models for Itaipú Power Plant using Modelica and OpenIPSL”. In: pp. 139–148. DOI: 10.3384/ecp20169139.

Qin, Yining et al. (2019-08). “A JModelica.org Library for Power Grid Dynamic Simulation with Wind Turbine Control”. In: *IEEE Power and Energy Society General Meeting*. Vol. 2019-Augus. IEEE, pp. 1–5. ISBN: 9781728119816.

Stott, Brian (1974). “Review of load-flow calculation methods”. In: *Proceedings of the IEEE* 62.7, pp. 916–929.

Tinney, William F and John W Walker (1967). “Direct solutions of sparse network equations by optimally ordered triangular factorization”. In: *Proceedings of the IEEE* 55.11, pp. 1801–1809.

Vanfretti, Luigi et al. (2017-04). “An open data repository and a data processing software toolset of an equivalent Nordic grid model matched to historical electricity market data”. In: *Data in Brief* 11, pp. 349–357. ISSN: 23523409.

Ward, J B and H W Hale (1956). “Digital computer solution of power-flow problems [includes discussion]”. In: *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems* 75.3, pp. 398–404.

Winkler, Dietmar (2017-09). “Electrical Power System Modelling in Modelica - Comparing Open-source Library Options”. In: pp. 263–270. DOI: 10.3384/ecp17138263.

Winkler, Dietmar (2019-02). “Analysing the stability of an Islanded hydro-electric power system”. In: pp. 103–111. DOI: 10.3384/ecp18154103.

Aircraft Mission Simulation with the updated FlightDynamics Library

Marc May[1] Reiko Müller[1] Gertjan Looye[1]

[1]DLR Institute of System Dynamics and Control, Oberpfaffenhofen, Germany,
marc.may@dlr.de, reiko.mueller@dlr.de, gertjan.looye@dlr.de

Abstract

Aircraft mission simulation plays an essential role during the simulative design process of aircraft systems. An important task is to conduct analyses for total aircraft assemblies in different flight phases and also to produce performance metrics in Multidisciplinary Design Optimization (MDO) loops. Based on DLR's FlightDynamics, the library presented in this paper introduces mission simulation capability for aerial and on-ground movement. For instance, this includes flight control and management functions, a detailed aircraft implementation including several subsystems (for example landing gears, actuation and sensor systems), and the application of the integrated setup to realistic use-cases.

Keywords: *Aircraft modeling, mission simulation, flight control.*

1 Introduction

During design and testing of aircraft and associated systems, the need for aircraft mission simulation arises in many occasions, for example to conduct MDO, validation and verification, trajectory optimization and prediction, as well as air traffic management. The main building blocks of a mission simulation setup are on the one hand the actual aircraft model with its various (sub-) systems, and on the other hand functions and algorithms (mostly in the form of controllers) enabling the aircraft to follow a mission plan (for example a gate to gate trajectory for a passenger aircraft). Concerning the first part, a well established tool to perform aircraft modeling in Modelica is DLR's FlightDynamics library (Looye et al. 2014), which has been used in a multitude of research projects so far. The second part, namely flight control and management algorithms allowing the guidance on ground and in the air was added to the FlightDynamics during the course of the EU-funded project OPeRATOR, which is associated with Systems Integrated Technology Demonstrator (SYS-ITD) of the CleanSky2 Joint Undertaking (MISSION Consortium 2015). Aside from DLR, partners of the OPeRATOR project (including a large aircraft supplier) seeked a library with this capability to help developing and assessing aircraft subsystems, for example novel actuator architectures.

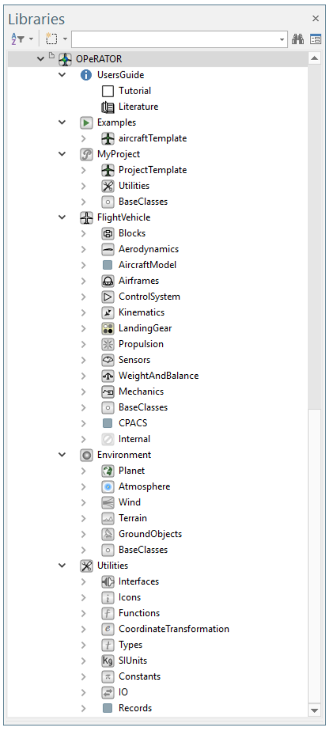
The mission simulation capability necessitated several adaptations and further development of the FlightDynamics library, which are laid out in this paper. The following section reviews the derived library structure, highlighting changes with respect to the original FlightDynamics. In Section 3, the modelling of the aircraft and associated systems will be explained, followed by the mission and scenario definitions in Section 4, where also the flight controller and flight management systems necessary to guide the aircraft on the mission are described. Results of the use-cases defined in the project for verification are shown in Section 5, and the conclusions are drawn in Section 6.

2 Library structure

As a derivative of the FlightDynamics library, OPeRATOR shares its high level library structure, which is shown in Figure 1. It furthermore adds a dedicated Controller package, which contains all mission planning and execution capabilites (i.e. controllers and flight management algorithms).

The first level subpackages will be shortly listed in the following. In ℹ UsersGuide, a text-based tutorial of the library, its components and modeling principle is given. Also, there is a list of references given in the Literature model. The ▶ Examples package implements the use-cases defined in the project, results of these are given in Section 5. 𝒫 MyProject is used to include and integrate new aircraft designs. Therefore basic building blocks needed to assemble a model, like aerodynamics, engines and systems are stated here, which represents a proposal how to structure a new aircraft implementation with new systems. ✈ FlightVehicle is the main location where all models and classes are stored. This comprises equations of motion in the kinematics package, implementations of aerodynamics (a tabulated dataset from Vortex Lattice Method (VLM) calculations is used in OPeRATOR, however arbitrary analytical and data-driven models are possible), engine and actuator models, sensors, landing gears, and so on. New models are added to the respective sub-library when they are designed. The ○ Environment package contains everything necessary for simulating the surrounding of the aircraft, for example geoid, gravity and

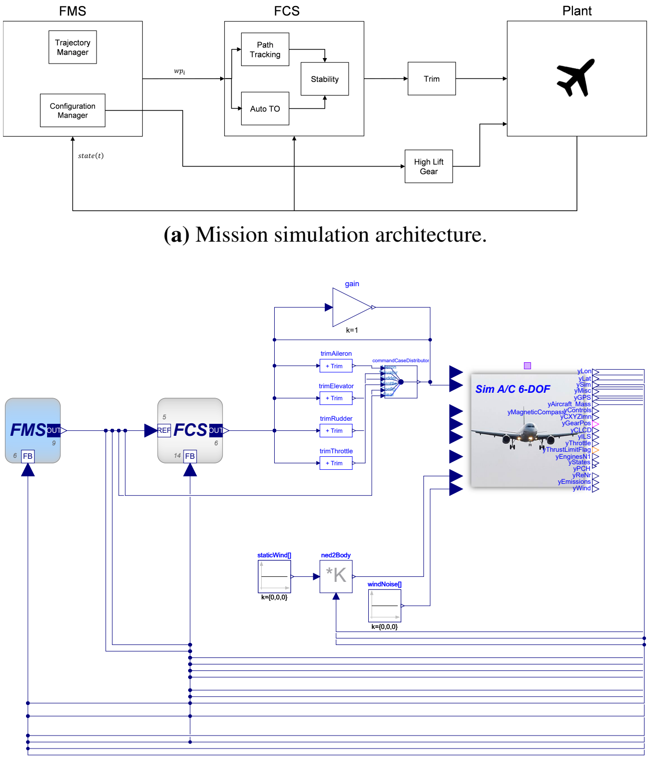
DOI
10.3384/ecp21181155



Libraries

- OPeRATOR
  - UsersGuide
    - Tutorial
    - Literature
  - Examples
    - aircraftTemplate
  - MyProject
    - ProjectTemplate
    - Utilities
    - BaseClasses
  - FlightVehicle
    - Blocks
    - Aerodynamics
    - AircraftModel
    - Airframes
    - ControlSystem
    - Kinematics
    - LandingGear
    - Propulsion
    - Sensors
    - WeightAndBalance
    - Mechanics
    - BaseClasses
    - CPACS
    - Internal
  - Environment
    - Planet
    - Atmosphere
    - Wind
    - Terrain
    - GroundObjects
    - BaseClasses
  - Utilities
    - Interfaces
    - Icons
    - Functions
    - CoordinateTransformation
    - Types
    - SIUnits
    - Constants
    - IO
    - Records

Figure 1. The library tree opened in SIMULATIONX® (this version contains a hidden controller package, which is mandated by export control regulations for dissemination to external partners).



(a) Mission simulation architecture.

(b) MODELICA implementation.

Figure 2. Layout of the flight management and control systems in conjunction with the aircraft model for mission simulation.

magnetic flux of the Earth, atmosphere and ground objects like localizer and glideslope emitters. The final package Utilities contains interface definitions, coordinate transformation and various helper functions and records. Finally, the Controller package collects all functionality concerning flight control. This encompasses autopilot functionality (e.g. acquiring and holding of reference altitude, speed and orientation), trajectory path tracking, controllers for stabilization and damping of aircraft eigenmodes and disturbances as well as controllers for specialized tasks like takeoff and landing.

3 Aircraft modeling

Depending on the use-case or application, an aircraft modeling library must provide different levels of fidelity or detail. In the case of aircraft mission simulation, reduced mass-point models were the standard approach so far, as they combine fast execution time and acceptable precision. Well-known implementations include for example the Base of Aircraft Data (BADA) model family (EUROCONTROL 2012) as a reduced longitudinal-plane aircraft performance model, and also inverse models with three Degrees of Freedom (DoFs), like they were used for instance in the DLR project TIVA (Liersch and Hepperle 2011). For the OPERATOR project, a six DoF rigid body aircraft model formulation was selected as a generic approach, which enables a more detailed simulation of transient flight, especially during turning.

The equations of motion (see e.g. Stevens, Lewis, and Johnson (2015) for a derivation) with introduction of forces and moments, as well as coordinate transformations are implemented using the MODELICA MULTIBODY library (Otter, Elmqvist, and Mattsson 2003), which allows straightforward graphical construction of airframe assemblies with components, like propulsion, actuation and sensor systems as well as payloads. Furthermore, the library employs partial models for all subcomponents based upon which different realisations of systems can be developed and exchanged very easily. Global models related to world/geodetics and atmosphere are adopted from the DLR ENVIRONMENT library (Briese, Klöckner, and Reiner 2017), and use the inner/outer concept of MODELICA. The overall architecture of the resulting mission simulation setup is shown in Figure 2, with some of the system components being described in the following sections, namely aerodynamics (3.1), actuators (3.2) and landing gears (3.3).

3.1 Aerodynamics

The aerodynamics model in the OPERATOR library works on tables previously generated by a multidisciplinary design and simulation network, for which the

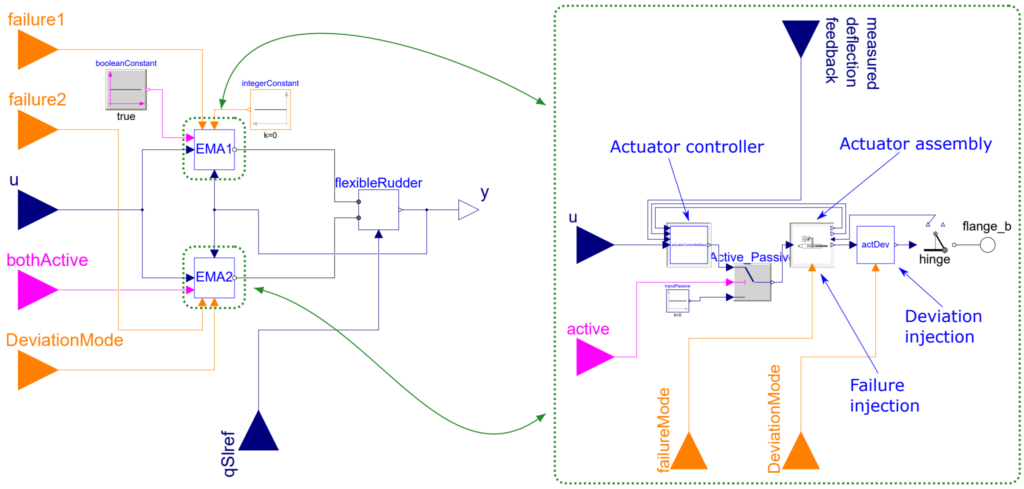


Figure 3. Aileron actuation system with redundant Electro-Mechanical Actuator (EMA) implementation.

MDO software Remote Computing Environment (RCE) (Seider 2014) is employed. The tools for each discipline of the multidisciplinary design are incorporated in RCE and are made available by several DLR institutes, each being specialized in a different area of aircraft design. The interconnected execution inside RCE allows design workflows adapted to the respective use-case (for example aircraft noise analysis and CFD/CSM simulation). The RCE workflow has already been used in different projects, for instance DIGITAL-X (Kroll et al. 2016) and VICTORIA (Görtz et al. 2020).

One of the results of the data calculated within RCE are aerodynamic tables, derived for example from Computational Fluid Dynamics (CFD) data. Regarding mission simulation and the large time horizons involved, the aerodynamics model has to balance the aspects of fidelity and precision versus computation time. To this end, for modeling of drag and lift contributions of the airframe and control surfaces, a fast VLM method (Hedman 1966) is combined with a lifting-line method (Horstmann 1986), yielding tables scheduled over different sets of variables from altitude, Mach and Reynolds numbers, angle of attack and sideslip angle, as well as body turn rates and control surface deflections.

As the scenario considers passenger aircraft trajectories, extreme maneuvering or conditions at the boundary of the flight envelope are not expected. This means that these linear methods are applicable and provide a good approximation of the aerodynamics during the entire mission. For modeling the increase in lift coefficient c_L due the ground effect, several empirical relations stated in Phillips and Hunsaker (2013) are implemented, which can be selected as a parameter in the aerodynamics model.

3.2 Actuation system

The actuation system can model physical actuators (for example by employing MULTIBODY library parts) as well as approximations like first and second order filters. Furthermore, a combination of approximated and physical actuators can be used. To demonstrate these functionalities, a use-case was defined regarding an actuator force-fight as introduced in more detail in Section 5.1. In this scenario, one aileron actuator is assumed to be defective leading to a force-fight with a second actuator which operates on the same flexible control surface. For this purpose, a redundant actuator setup with two Electro-Mechanical Actuators (EMAs) per control surface was developed as shown in Figure 3. The deviations are adjusted via an integer variable that triggers the following cases:

- Gain difference in the actuators, which leads to divergence in positions and rates.
- Time delay, meaning that the actuators react at different time instants.
- Position offset, i.e. the actuators extend to different positions when subjected to the same command input.

In a similar way, it is possible to introduce failures of the actuators, such as a stuck actuator or actuator runaway with a parameterized rate up to a maximum / minimum deflection.

3.3 Landing gear

The aircraft's undercarriage is represented by a simplified landing gear model assembly consisting of two main and one nose gear (see Figure 4). All gears are suspended by a spring-damper element. The two main gears feature brakes, while the nose gear is not braked and has a steering. By means of a MULTIBODY frame connector (Otter, Elmqvist, and Mattsson 2003), the gear is attached to the airframe. In order to facilitate numerical calculation of tyre friction and general forces and moments of

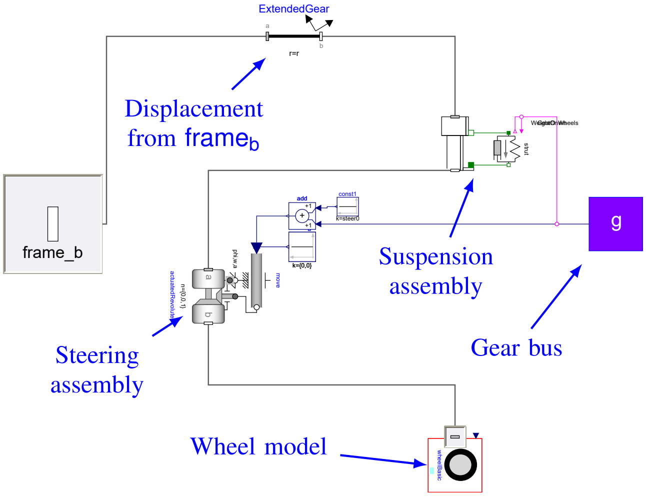


Figure 4. Steerable nose gear implementation. The connector frame_b is attached to the aircraft reference point in the top-level airframe assembly.

the gear assembly, the wheels are defined to remain on the ground-projected local coordinate system below the aircraft during the complete flight. Forces and moments are only introduced to the system if the landing gear assembly touches the ground as represented by the altitude-triggered boolean variable `WeightOnWheels`. A tyre friction model has been adopted from Otter, Elmqvist, and Mattsson (1999), which considers a hybrid formulation for the different states (forward- and backward rolling, stuck, sliding) for Coulomb friction in x- and y-direction of the tyre local coordinate system. If the friction elements are dynamically coupled, event definition for these states leads to a mixed continuous/discrete system of equations, that has to be solved by the numerical integration algorithm. Finally, an empirical roll-coefficient model according to Barnes and Yager (1998) has been implemented which allows to consider different runway surface conditions (dry, wet, flooded, icy, snow-covered) depending on loading, tyre pressure, braking threshold and external conditions. This overall approach allows to efficiently handle the unavoidable event iterations for the discrete landing gear touchdown, and provides good estimates for the forces and moments involved.

3.4 Model validation and verification

As OPERATOR is based on the FLIGHTDYNAMICS library, several components have been validated and verified during its commercialization process (for example the environment models, kinematics, aerodynamics, engine, weight and balance, actuator, sensor, terrain and wind modules as well as interfaces). Nevertheless, model verification was performed at hand of five use cases (controlled flight in cruise, actuator force fight, automatic landing, rejected takeoff, and full city-pair mission) in OPERATOR. Data like aerodynamics and propulsion maps were generated by verified processes and tools within the DLR by the institutes specialized in the respective topics (for example the Institute of Propulsion Technology[1]). In CLEANSKY2, the OPERATOR library is also employed by industrial partners (e.g. Collins Aerospace) in combination with their own proprietary models, where additional validation and verification activities are performed accordingly.

4 Flight management and control

It is assumed in the OPERATOR project, that the general mission definition is stated beforehand and subsequently provided to the mission simulation tool. The respective input file specifies a gate-to-gate mission separated into several phases, as it is most common practice in Air Traffic Management (ATM) and flight planning.

Table 1. Mission simulation input data.

Mission point ID	Segment duration [s]	Altitude [ft]	Mach number
Ramp Up	900	0	0
Taxi	900	0	0.025
Takeoff	60	0	0.2
Climb	180	4000	0.4
Level flight	1020	39000	0.6
Level flight	2700	39000	0.78
Descent	1320	2350	0.6
Approach and flare	120	0	0.25
Landing	60	0	0.025
Taxi	480	0	0.025

4.1 Mission definition

Within the phases, values for several variables are defined which parameterize the segment (see Table 1) and which may also serve as parameters for an outer optimization loop. According to these definitions, a variable denoting the current flight state is synthesized, which in turn controls the modes and settings in the Flight Management System (FMS). In MODELICA, this is realized by the enumeration `FlightState`, which can assume the values `UN` – Undefined (this is the initial state, from which transition to other states can happen, see Figure 5), `RU` – Aircraft Ramp Up, `TA` – Taxi phase, `TO` – Takeoff phase, `CL` – Climb phase, `CR` – Cruise phase, `AP` – Approach phase and `LD` – Landing phase.

The switching between flight states is realized with a state machine (see Figure 5) and depends on several boolean conditions (grey source blocks) which trigger the transitions between states. The conditions are determined by variables such as the weight on wheels, the distance

[1] `www.dlr.de/at/en/`

to target runway, altitude-, orientation-, rates, airspeed and corresponding limits. The simulation is usually terminated if the landing is completed and the aircraft has stopped, but other exit conditions can be defined easily.

4.2 Flight management

To discretize the trajectory with the `FlightState` variable, it needs to be processed initially. In Figure 6, the overall process of generating an FMS solution from the textual mission input data using the Trajectory manager is shown. By taking into account the target runway position (latitude φ_{rwy}, longitude λ_{rwy}, altitude h_{rwy}) and heading χ_{rwy} as well as tabulated time intervals Δt, altitudes h and velocity/Mach number Ma, the trajectory is back-propagated in position starting at the touchdown point. During this initialization, the initial trajectory point is moved into the touchdown point, and direction is rotated by 180°. A loop over all segments is performed, which allows to calculate the positions of transition points (also in time) and the ground distance (on great circle paths) between points. This finally yields the departure conditions in position and initial heading. The trajectory is hence represented by linear segments where additional transient phases are considered to allow smooth transition between segments and flight states. The user can influence these transitions by specifying maximum accelerations in longitudinal and vertical directions. By taking into account the state feedback x_{feedback}, continuous commands in the variables altitude h_{cmd}, pitch angle γ_{cmd}, inertial velocity V_{cmd}, course angle χ_{cmd} and sideslip angle β_{cmd} are provided for the current flight state.

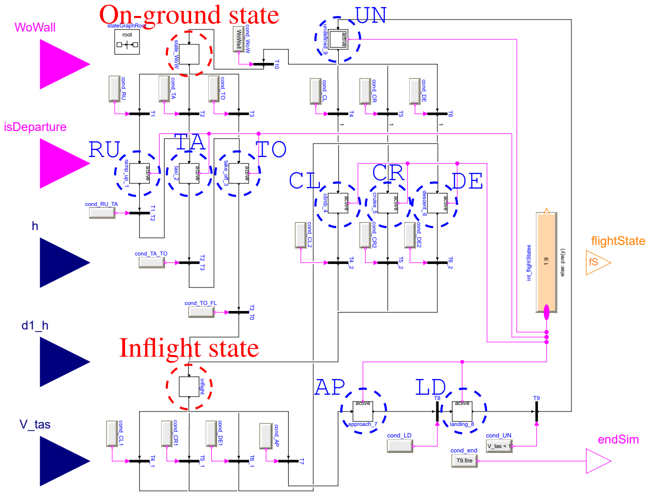


Figure 5. The state machine for switching of flight states using the MODELICA STATEGRAPH library. From the initial `UN` state, it can transition either to on-ground (`RU`, `TA`, `TO`) or inflight-states (`CL`, `CR`, `DE`).

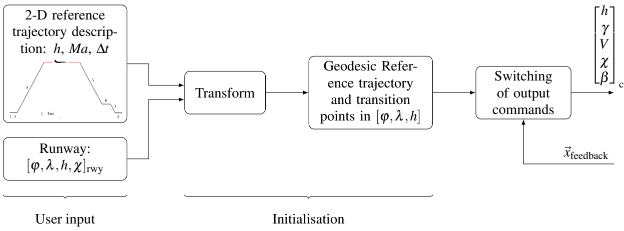

2-D reference trajectory description: h, Ma, Δt
Runway: [φ, λ, h, χ]rwy
Transform
Geodesic Reference trajectory and transition points in [φ, λ, h]
Switching of output commands
[h γ V χ β] cmd
x⃗feedback
User input
Initialisation

Figure 6. Transformation of mission input data using the Trajectory manager.

4.3 Flight controller

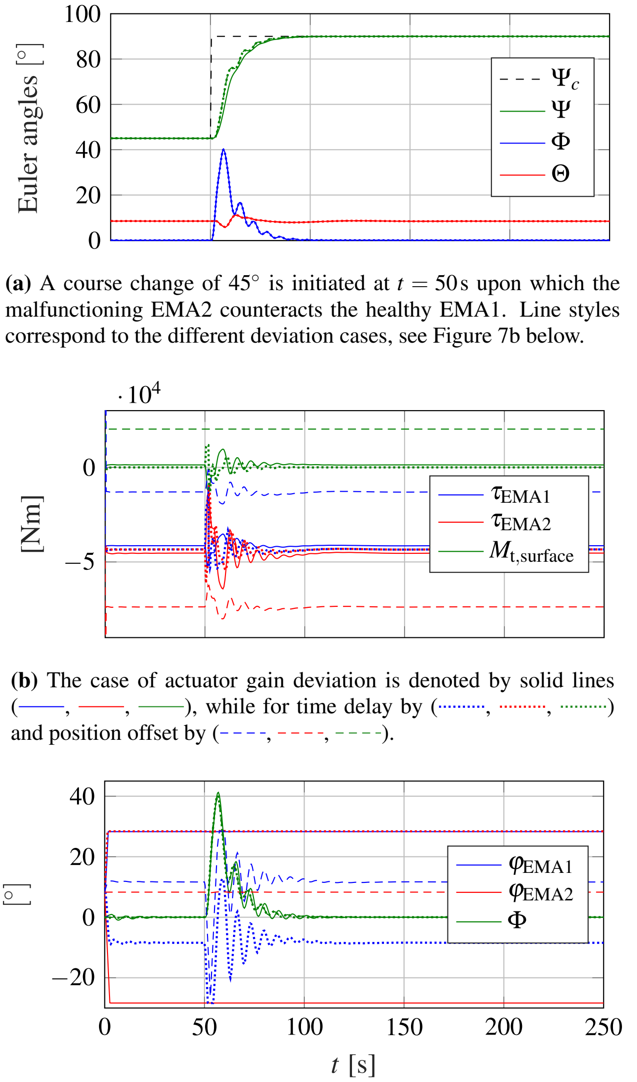
For the Flight Control System (FCS), the common cascaded controller structure with autopilot outer- and stabilization inner loop is adopted (see Figure 2a). The path tracking module receives the aforementioned continuous commands from the FMS and contains modes for inflight and on-ground movement, while a dedicated block is developed for automatic takeoff. These two modules form the autopilot part of the Flight Control System (FCS), which issues aircraft orientation and throttle commands to the stabilization loop. Automatic landing is performed with the guidance by Instrument Landing System (ILS) and Localizer sensor models, which provide lateral and vertical deviations to the glideslope controller. It also contains a mode for the landing flare before touchdown, which works according to the variable τ - law described in Lambregts (1982).

5 Simulation examples

In this section, the results of the use-cases defined in the OPERATOR project are presented. The use-cases include the force-fighting scenario in Section 5.1, the rejected takeoff scenario in Section 5.2, and the complete mission in Section 5.3. All scenarios have been tested with SIMULATION-X® version 4.1 and DYMOLA® 2018 FD01 on a standard PC-workstation (INTEL XEON E5-1630 v3, 16 GB RAM) with WINDOWS 10. The real time factor in DYMOLA® is varying (force-fight: 328.9, rejected takeoff: 26.4, mission: 213.5) for this configuration and the considered scenarios, which is due to different initialization states (initialization on ground with extended landing gear, or with extended actuator model is more complicated and takes more time), the number of states and the system stiffness.

5.1 Actuator force fight

The inflight force-fighting scenario has the goal of evaluating the aircraft's behaviour with respect to control inputs and possible faults. To this end, the implementation considers an actuator force fight scenario, where redundant aileron actuators allow to model different types of force fights. These are possible differences in actuator gain, and time delay, as well as deflection offsets which results in deformation of the flexible control surface.



(a) A course change of 45° is initiated at $t = 50$ s upon which the malfunctioning EMA2 counteracts the healthy EMA1. Line styles correspond to the different deviation cases, see Figure 7b below.

(b) The case of actuator gain deviation is denoted by solid lines (——, ——, ——), while for time delay by (·········, ·········, ·········) and position offset by (- - - -, - - - -, - - - -).

(c) .The actuator failure cases are denoted by (——, ——, ——) for an upwards runaway, (·········, ·········, ·········) for a downwards runaway and a stuck actuator is indicated by (- - - -, - - - -, - - - -).

Figure 7. Results of the actuator force fight simulation.

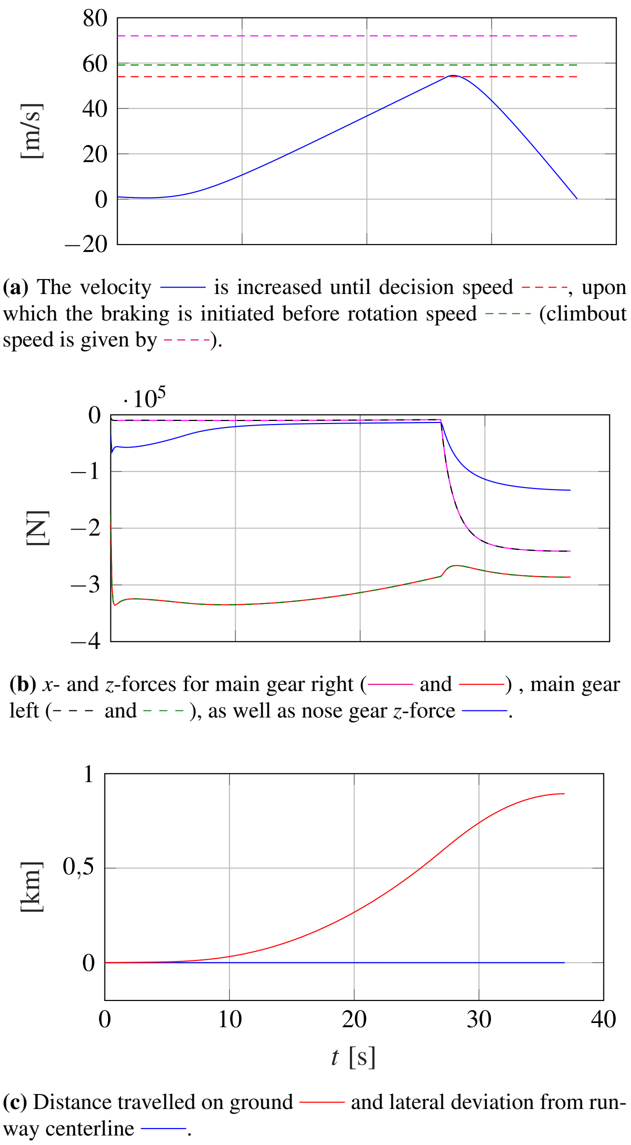
Results for deviations of the actuator gain of +25%, a time delay of 0.2 s and a 10 cm actuator rod position offset are shown in Figure 7b, where always the second actuator (EMA2) introduces the deviation. During a course change (in this case 45°), the aircraft has to build up a roll angle for turning flight and reduce it again to return to wings level (the roll angle limit is set to ±45° inside the lateral autopilot). The actuators twist the flexible aileron control surface via torques τ_{EMA1}, τ_{EMA2} and hence introduce a torsional moment $M_{\text{t,surface}}$ (indicated by green color) that is different for the three deviation types. The difference in the overall gain of the actuator controller for instance produces a nonzero twisting moment (——), while the moment due to a time delay (·········) vanishes if there is no control activity (i.e. when the new course is attained). The position offset in EMA2 - - - - is introduced after initialization/trimming in order to make the three solutions comparable. EMA1 - - - - has to produce a countering torque (symmetrical to the trim value, which is approximately represented by ········· and ·········), which leads to a large twisting torque - - - - in the control surface.

In Figure 7c, simulation results for three actuator failure types are depicted. It has to be noted that the aileron control surface can deflect upwards with larger angles than downwards before reaching limits/stops. Hence runaways of EMA2 with a rate of 0.25 m/s show, that for the downwards direction EMA1 ········· compensates the runaway of EMA2 ········· and still has control authority left during the turn at $t = 50$ s. This cannot be accomplished in the upwards runaway case, as due to the smaller downward deflection, EMA1 —— cannot cancel EMA2 —— anymore and therefore does not contribute to turning and also needs compensating moments from other aileron control surfaces. The final stuck case is similar to the downwards case, as EMA2 - - - - remains at a relatively low deflection, which can be cancelled by EMA1 - - - -. As can be seen in Figures 7a and 7c, the resulting Euler angles are very similar, with only small differences in the roll angle Φ and heading Ψ.

5.2 Rejected takeoff

The rejected takeoff is a certification relevant test where the aircraft is required to stop in a certain amount of distance while the takeoff procedure is already underway. Due to the high speeds of around 300 km/h, a large amount of kinetic energy has to be converted to heat by the braking system, additionally in a short amount of time and taking into account the limited runway length. Depending on these factors and also parameters like ground roll coefficients, payload, on-ground controller etc., the decision speed beyond which no safe stopping is guaranteed, varies accordingly.

Representing an emergency maneuver, a dedicated braking mode is activated in the controller, while standard FMS and longitudinal FCS loops are switched off. This mode issues brake and reverse throttle commands (if supported by the engine model) to minimize the braking distance. In the simulation, this is triggered if the on-ground velocity surpasses the decision speed (this marks the point where in the case of problems the aircraft shall still be able to stop safely without taking off). To ensure smooth transitions, the velocity (0 m/s) and braking commands (100%) are filtered before being passed to the FCS. The lateral on-ground mode also acts to keep the aircraft at the centerline via the lateral autopilot and inner loops commanding the rudder and nose-wheel steering with a speed-variable gain (see Figure 8c). In the second subplot, the resulting longitudinal and vertical forces on the tyres of the nose and main gears are shown. As the aircraft accelerates, the nose wheel is unloaded (——) and the main gears are loaded in z-direction (—— and - - -) due to the pitch-up moment, in-



(a) The velocity —— is increased until decision speed - - - -, upon which the braking is initiated before rotation speed - - - - (climbout speed is given by - - - -).

(b) x- and z-forces for main gear right (—— and ——) , main gear left (- - - and - - -), as well as nose gear z-force ——.

(c) Distance travelled on ground —— and lateral deviation from runway centerline ——.

Figure 8. Results of the rejected takeoff simulation.

duced by the engine to Center of Gravity (CoG) moment arm. Conversely, the main gear x-forces (—— and - - -) increase when brakes are applied, and the nose gear z-force increases due to the pitch-down moment during braking. Note that the z-values do not return to their stationary values, as the simulation is terminated as soon as the velocity falls below 0.1 metres per second. With a total mass of 72.69 t, the aircraft is able to stop after a distance of 882.5 metres (——), including the acceleration phase.

5.3 City pair mission

As described in Section 4, a standard city-pair mission is generated from the data supplied by the user. This includes climb, cruise and descent phases, which are flown in the longitudinal plane, i.e. except for departure and approach, there is no lateral manoeuvring. In the simulation and optimization studies targeted by the MISSION

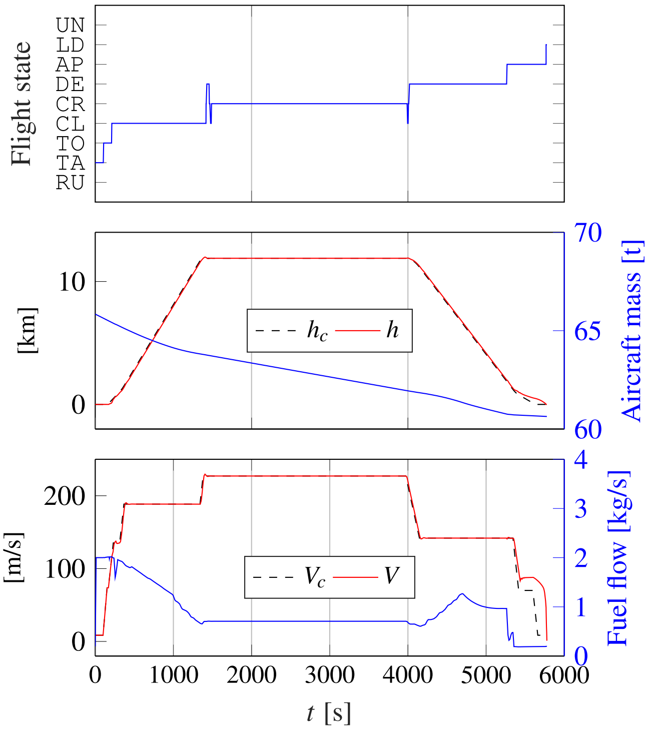


Figure 9. Results of the mission simulation.

project, the aircraft has to climb for instance from ground altitude to cruise altitude and can do so in a variety of trajectories that are optimal for the functioning of specific systems and the mission length. When introducing additional system models, such as an Environmental Control System (ECS), the effects of environmental conditions (e.g. temperature, humidity) on dedicated aircraft systems can be studied. Similarly, during descent, the aircraft has to descend from cruise altitude to approach altitude, and the (auto-)pilot has to reduce the engine throttle. Again studies regarding ECS and engine operational modes can be conducted.

These prerequisites motivate the presented longitudinal approach in the main flight phases and a full 3-D simulation during departure and approach, for example to correctly simulate an ILS guidance. In Figure 9, the commanded and actual altitude h_c and h as well as velocities V_c and V are shown. Due to the flight controller, the aircraft can follow the reference trajectory with only small deviations. One exception is during approach, where the mission definition commands a trajectory that is higher and faster than what the aircraft is capable to achieve. The controller adjusts the commands so that the throttle is reduced to idle (see fuel flow) and the descent rate is at the current maximum value. The final approach is performed using the ILS system and flare controller as described in Section 4.3.

6 Conclusion and outlook

In this paper, recent extensions to the FLIGHTDYNAMICS library enabling passenger aircraft mission simulation and specialized case studies have been presented. Through integrated flight management and control systems, it is possible to conduct a variety of simulation studies, for example to design and evaluate subsystems like actuators, controllers, Environmental Control Systems (ECSs), landing gear systems and so on. This is showcased by means of three examples, an inflight actuator force-fight, a rejected takeoff simulation and a complete city-pair mission scenario.

In the future, the library will be continuously extended and employed for further studies and developments at the DLR, for example in the fields of unmanned and autonomous flight, simulation of future propulsion concepts (electrical, hydrogen, synfuels) as well as design and validation of control systems.

Acknowledgements

The work in project OPERATOR was funded by the CLEANSKY2 joint undertaking under topic number JTI-CS2-2017-CfP07-SYS-03-16.

References

Barnes, Arthur G. and Thomas J. Yager (1998). “Enhancement of aircraft ground handling simulation capability”. In: *Advisory Group for Aerospace Research and Development (AGARD).*

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, Prague, Czech Republic, May 15-17, 2017.* 1 132. Linköping University Electronic Press, pp. 929–938.

EUROCONTROL (2012). *User manual for the Base of Aircraft Data (BADA) Revision 3.10.* Technical/Scientific Report No. 12/04/10-45. EEC.

Görtz, Stefan et al. (2020-09). “Ergebnisse des DLR-Projekts VicToria - Virtual Aircraft Technology Integration Platform”. In: *Deutscher Luft- und Raumfahrtkongress 2020 (DLRK2020).* URL: https://elib.dlr.de/135959/.

Hedman, Sven G. (1966). *Vortex lattice method for calculation of quasi steady state loadings on thin elastic wings in subsonic flow.* Tech. rep. DTIC Document.

Horstmann, Karl-Heinz (1986). “Ein Mehrfach-Traglinienverfahren und seine Verwendung für Entwurf und Nachrechnung nichtplanarer Flügelanordnungen”. PhD thesis. TU Braunschweig.

Kroll, N. et al. (2016-03). “DLR project Digital-X: towards virtual aircraft design and flight testing based on high-fidelity methods”. In: *CEAS Aeronautical Journal* 7.1, pp. 3–27. ISSN: 1869-5590. DOI: 10.1007/s13272-015-0179-7.

Lambregts, Antonius A. (1982). “Avoiding the pitfalls in automatic landing control system design”. In: *AIAA Guidance and Control Conference.* American Institute of Aeronautics and Astronautics. Chap. 6, pp. 799–809. DOI: 10.2514/6.1982-1599.

Liersch, Carsten M. and Martin Hepperle (2011-12). “A distributed toolbox for multidisciplinary preliminary aircraft design”. In: *CEAS Aeronautical Journal.* CEAS Aeronautical Journal Volume 2.Number 1-4, pp. 57–68. URL: https://elib.dlr.de/74509/.

Looye, Gertjan et al. (2014-09). “Object-Oriented Aircraft Modelling with the DLR FlightDynamics Library”. In: *63. Deutscher Luft- und Raumfahrtkongress.* Augsburg, Germany: Deutsche Gesellschaft für Luft- und Raumfahrt.

MISSION Consortium (2015). *MISSION Requirements Document and Exploitation Plan, Clean Sky 2 SYS-ITD Deliverable, D100.3.1.2.*

Otter, Martin, Hilding Elmqvist, and Sven Erik Mattsson (1999-09). “Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle”. In: *IEEE International Symposium on Computer Aided Control System Design*, pp. 151–157. DOI: 10.1109/CACSD.1999.808640.

Otter, Martin, Hilding Elmqvist, and Sven Erik Mattsson (2003). “The new Modelica Multibody Library”. In: *3rd International Modelica Conference*, pp. 311–330.

Phillips, W. F. and D. F. Hunsaker (2013). “Lifting-Line Predictions for Induced Drag and Lift in Ground Effect”. In: *Journal of Aircraft* 50.4, pp. 1226–1233. DOI: 10.2514/1.C032152.

Seider, Doreen (2014-11). “Open Source Framework RCE: Integration, Automation, Collaboration”. In: *4th Symposium on Collaboration in Aircraft Design.* URL: https://elib.dlr.de/93323/.

Stevens, Brian L., Frank L. Lewis, and Eric N. Johnson (2015). *Aircraft control and simulation: dynamics, controls design, and autonomous systems.* John Wiley & Sons.

Modelica-Based Modeling on LEO Satellite Constellation

Chan Liu[1] Yikai Qian[2] Liping Chen[1] Yan Qu[3] Fanli Zhou[3]

[1]School of Mechanical Science and Engineering, Huazhong University of Science and Technology, China,
{liuchan,chenlp}@hust.edu.cn
[2]Shanghai Institute of Space Propulsion, China, jt0033063@163.com
[3]Suzhou Tongyuan Software&Control Tech. Co. Ltd, China, {quy,zhoufl}@tongyuan.cc

Abstract

The new generation of LEO (Low Earth Orbit) communication satellite constellations have the advantages of low latency, strong signals, and global coverage. In order to achieve global coverage, the LEO satellite constellations are often very large, with a large number of satellites, which puts forward high requirements for the overall design, operation and maintenance of the constellation. Based on the Modelica language, this paper conducts a detailed study of the LEO communication satellite constellation and built a system model of satellite constellation, and carries out a full-link simulation for the typical service of mobile communication. By analyzing the simulation results, we conclude that based on the model of satellite constellation proposed in the paper, the satellite constellation can be quickly designed and simulated.

Keywords: satellite constellation, Modelica, satellite communication, full-link simulation

1 Introduction

In recent years, as the demand for network communication has increased sharply, the idea of using LEO communication satellite constellation to provide the Internet from space has become popular again due to its low communication delay and full-area coverage. A group of LEO communication satellite constellation projects represented by OneWeb, StarLink and Iridium II have emerged in the world. Compared with HEO(High Earth Orbit), LEO satellites have a shorter lifespan, and the technical updates and iterations of the constellation system are more frequent. The existing simulation methods for satellite constellations can be mainly divided into the following two types: The first is to simulate the orbit, perturbation, structure and ground coverage characteristics of the satellite constellation based on software such as STK(Q. Wang and Xie 2021), and to support the constellation orbit and structure design; the second is based on software like OPNET(C. Hu and S. Wang 2018; Zhang 2008) to simulate the communication process of paging, inter-satellite link establishment, link switching, delay and other communication processes in the communication satellite constellation to calculate the communication link performance, thereby supporting the constellation network topology and network routing algorithm design. The two types of simulation methods are respectively aimed at constellation orbit design and communication link design. Currently, there is a lack of simulation methods that can unify the two calculations.

The core content of the paper is the construction of a multi-domain unified model for the satellite constellation system. The system model established by the Modelica language covers the ground segment, space segment and user segment equipment models. In the simulation application of mobile communication services, it can not only reflect system-level features such as constellation orbit, constellation structure, ground coverage, ISL(inter-satellite link), and satellite-to-earth links, can also simulate the energy balance, attitude control, fuel margin, and antenna gain of each satellite. The following section covers the introduction to the structure and characteristics of the LEO communication satellite constellation system. Section 3 mainly introduces the architecture of satellite constellation model and the principle of the main component model. Section 4 analyzes the simulation results of system model given example parameters. Section 5 gives conclusions.

2 Principle of LEO Satellite Constellation

LEO communication satellite constellation refers to a constellation system distributed at an orbital height of 700km to 2000km, which is usually divided into three components: space segment, ground segment, and user segment, as shown in Figure 1. Among them, the space segment refers to all satellites in the constellation; the ground segment usually includes the customs station, system control center and ground integrated network, responsible for the space segment satellite monitoring and control, and the operation and management of the space network; the user segment refers to various user terminals, including mobile terminals, shipboard terminals, vehicle terminals, and airborne terminals, etc.

There are mainly three types of links in the entire constellation system, feeder links, ISL and user links. Both the feeder link and the user link belong to the satellite-to-earth link, but the feeder link connects satellite and ground station, and the user link connects satellite and user terminal. The ISL refers to the link established between any

DOI
10.3384/ecp21181163

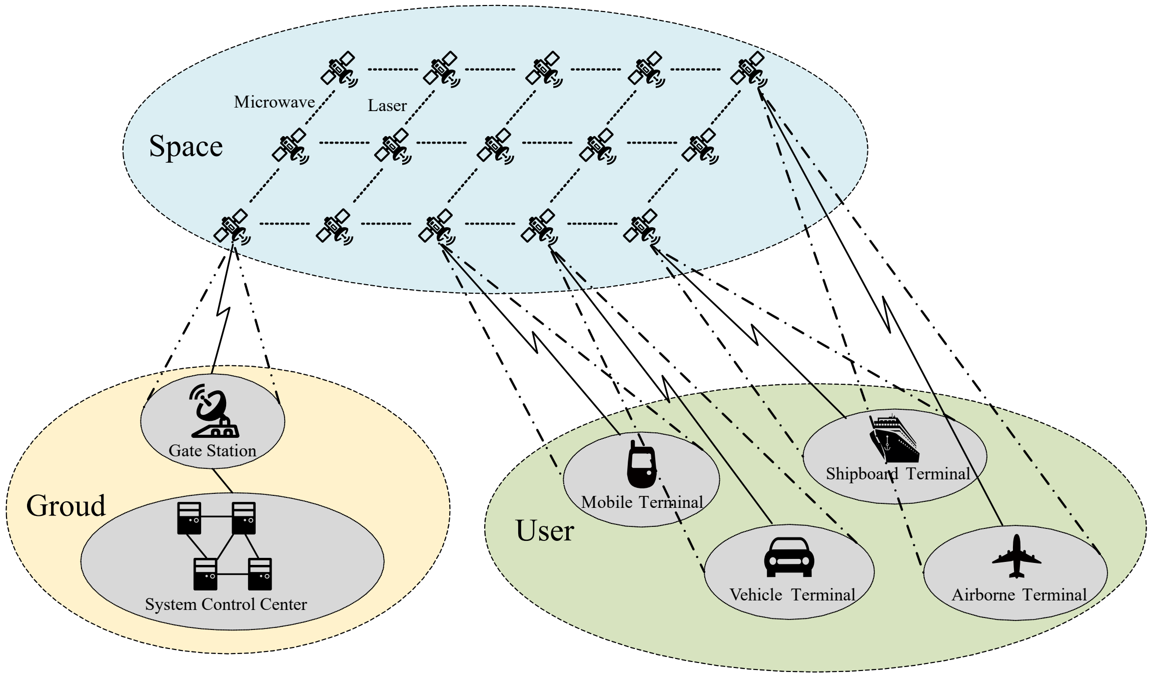


Figure 1. Schematic of a LEO communication satellite constellation.

satellite in the constellation and two adjacent satellites in the same orbital plane or between two adjacent satellites in a different orbital plane, which will occur with the switching of user links Change, the establishment of ISL generally follow the minimum hop number rule or the shortest propagation path rule.

For LEO communication satellite constellation system, the existing simulation methods are mainly based on different simulation platforms to simulate the constellation orbit coverage and communication link characteristics. Although this method can also achieve the purpose of simulation, it is not friendly for designers. And simulation tools such as STK or OPNET can only focus on system-level performance, and cannot considering the multi-domain characteristics of the satellite itself in the satellite constellation. This article uses Modelica language to establish a LEO communication satellite constellation system model. The model framework will be described in Section 3. Simulation based on this model can not only unify the constellation orbit coverage simulation and the constellation communication link simulation, but also focus on the operating status of a single satellite in the constellation system. This method not only improves the design efficiency of designers, but also facilitates the operation and maintenance of the constellation system. Simulation content is as follows:

- Constellation orbit and coverage characteristics, mainly including the position and speed of each satellite in the constellation at any time, and the overall ground coverage of the constellation.
- Communication link characteristics include two aspects, one is the margin for establishing communication links in any two places on the ground, and the other is the establishment and handover of ISL.
- Multi-domain characteristics of satellite, mainly including energy balance calculation, attitude control and orbit control maneuvers, and propellant margin calculation.

3 Unified Modeling of LEO Satellite Constellation

This section will propose a unified model architecture for constellation orbit coverage simulation, communication link simulation and satellite multi-domain simulation and introduce in detail the components of LEO satellite constellation developed by MWorks/Modelica.

3.1 Model Architecture

In order to realize the unified model of various simulation tasks of constellation, the constellation system model architecture is proposed as shown in Figure 2. Each color line with arrows in the figure represents a specific interface. The constellation orbit model calculates the number of orbital elements of each satellite in the constellation according to the given constellation design parameters. These orbital elements contain the position and speed information of the satellite and are transmitted to the ground station through the satellite-to-ground link. When user terminal 1 sends a request for establishing communication with terminal 2 to ground station, the communication request contains communication type, and the latitude

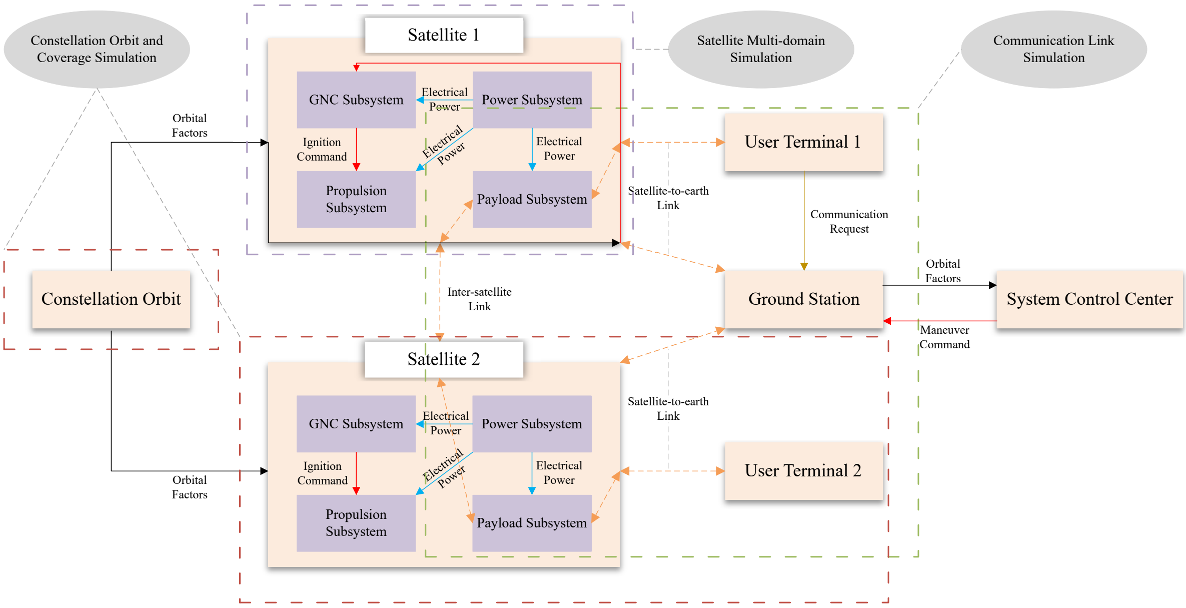


Figure 2. Model architecture of LEO communication satellite constellation system.

and longitude information of terminal 1 and terminal 2. Then ground station can establish a communication link between the two places, and calculate the link margin.

The link interface contains communication frequency, transmission rate, connection status, communication data, etc. The system control center generates the satellite orbit maneuvering control signal and transmits it to the GNC subsystem of the satellite that needs to maneuver through the satellite-to-ground link. The GNC subsystem then generates the propeller ignition command to the propulsion subsystem. With the help of the flexible component interface definition method of Modelica language, the modeling of related components can be easily carried out.

3.2 Model Implementation

Orbit, satellite and system control center are the three most important parts of LEO satellite constellation system, so this section mainly introduces the model realization of these three parts.

3.2.1 Orbit Model

The main parameters of the ideal constellation orbit model are: the number of orbits in the constellation p, the number of satellites in each orbit plane s, and orbital factors (semi-major axis a, eccentricity e, inclination i, RAAN(right ascension of ascending node) , argument of perigee and true anomaly f). The number of orbits and the number of satellites in each orbit plane determine the scale of the constellation. The semi-major axis and eccentricity of the orbit determine the size and shape of the orbit. The inclination, RAAN, and argument of perigee determine the orientation and direction of the orbit in space. The true anomaly determines the position of the satellite. Based on the above parameters, the coordinates(x,y,z) of each satellite in the geocentric inertial system can be calculated by Equation 1, so as to simulate the orbital state of the constellation at any time under ideal conditions.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \frac{a(1-e^2)}{1+e\cos f} \cdot \begin{bmatrix} \cos\Omega\cos(\omega+f) - \sin\Omega\sin(\omega+f)\cos i \\ \sin\Omega\cos(\omega+f) - \cos\Omega\sin(\omega+f)\cos i \\ \sin(\omega+f)\sin i \end{bmatrix} \tag{1}$$

In order to simulate the state of satellite orbits in space more realistically, orbital perturbations(Chen and Lin 2020) including aspheric gravity, atmospheric drag, sun-moon gravity, sunlight pressure, and post-Newton effect are introduced, as shown in Figure 3.

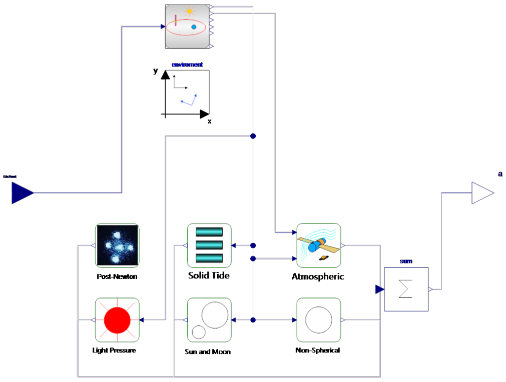


Figure 3. Orbit perturbation model.

- Non-spherical gravitational perturbation.

The Earth's gravitational field model adopts 70x70 Joint Gravity Model 3 (JCM3). The calculation of Non-spherical gravitational perturbation acceleration projected to the spherical coordinate component in the ground-fixed coordinate system is as follow:

$$\begin{aligned} a_N = & -\frac{\mu}{r^2\cos\varphi}\sum_{n=2}^{\infty}\sum_{m=0}^{n}(\frac{R_e}{r})^n\{(1+n)\cos\varphi \\ & \overline{\mathbf{P}}_{\mathbf{nm}}(\sin\varphi)(\overline{C}_{nm}\cos m\lambda+\overline{S}_{nm}\sin m\lambda)e_r \\ & [n\sin\varphi\overline{\mathbf{P}}_{\mathbf{nm}}(\sin\varphi)-N_n m\overline{\mathbf{P}}_{\mathbf{n-1,m}}(\sin\varphi)] \\ & (\overline{C}_{nm}\cos m\lambda+\overline{S}_{nm}\sin m\lambda)e_\varphi+m\cdot \\ & (\overline{C}_{nm}\sin m\lambda-\overline{S}_{nm}\cos m\lambda)e_\lambda\} \end{aligned} \tag{2}$$

where e_r, e_φ and e_λ are the three orthogonal unit vectors of spherical coordinates. λ and φ denote geocentric longitude and geocentric latitude. $\overline{C}_{nm}$ and $\overline{S}_{nm}$ are the normalized gravitational coefficient. $N_{nm} = \sqrt{\frac{2n+1}{2n-1}(n+m)(n-m)}$. $\overline{\mathbf{P}}_{\mathbf{nm}}(\mathbf{u})$ is the normalized gravitational coefficient. n represents the truncation order.

- Atmospheric drag perturbation.

The acceleration of atmospheric drag on a satellite with an area-to-mass ratio is:

$$a_D = -\frac{1}{2}C_D\frac{S}{m}\rho|V|\cdot V \tag{3}$$

where C_D is damping coefficient. ρ is atmospheric density at the location of the satellite. V is the speed of the satellite relative to the atmosphere.

- Lunisolar gravitational perturbation.

The perturbation acceleration of the sun and the moon to the satellite can be expressed as:

$$a_T = -\mu_s(\frac{r-r_s}{||r-r_s||^3}+\frac{r_s}{r_s^3}) \tag{4}$$

where r and r_s are the position vector of the satellite and the sun or moon in the geocentric inertial system. μ_s is gravitational coefficient.

- Solar radiation perturbation.

The perturbation acceleration caused by solar radiation on the satellite can be expressed as:

$$a_R = KC_R\frac{S}{R}\frac{L_s}{4\pi c}\frac{r-r_s}{||r-r_s||^3} \tag{5}$$

where C_R is solar radiation coefficient. c is Speed of light. L_s represents luminosity of the sun. K is solar visibility coefficient.

- post-Newtonian effects.

The perturbation acceleration produced by the first-order post-Newtonian effect term is:

$$a_P N = \frac{\mu}{c^2r^3}[(4\frac{\mu}{r}-v^2)\dot{r}+(4\dot{v}\cdot\dot{r})\dot{v}] \tag{6}$$

where $\dot{r}$ and $\dot{v}$ are satellite position vector and velocity vector.

3.2.2 Satellite Model

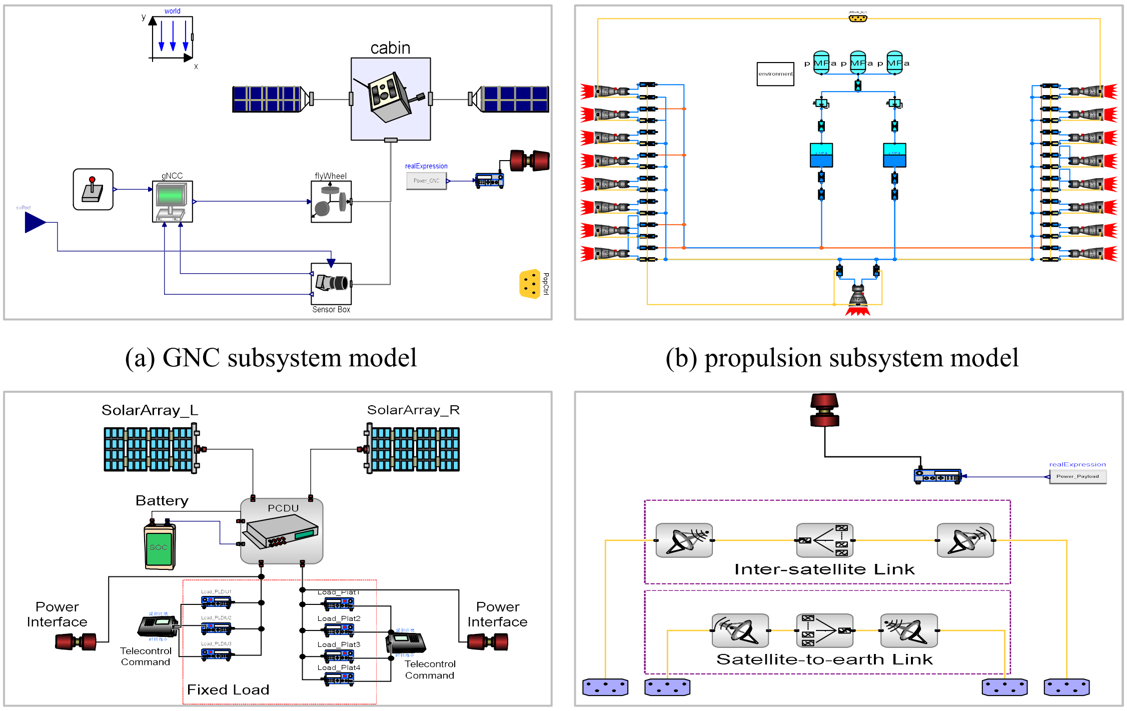
The satellite model is composed of four subsystem models, namely GNC subsystem model, propulsion subsystem model, payload subsystem model and power subsystem model.

Among them, the GNC subsystem consists of a satellite body model, a solar wing sail model, a sensor model, a flywheel model and a controller model, as shown in Figure 4(a), which is mainly responsible for the attitude and orbit control of the satellite. In order to maintain the satellite's payload to orient or track a specific target, attitude control is required. There are two ways to achieve attitude control. Small-scale attitude control is achieved by the controller generating a control signal to the flywheel, while large-scale attitude control is realized by the controller generating the switch signal of the attitude control engine.

The main task of the propulsion subsystem is to cooperate with the GNC subsystem to complete the attitude and orbit control during the life of the satellite. According to the function, it can be divided into a gas pressurization module, a propellant storage module and a thruster unit module, as shown in Figure 4(b). The gas pressurization module is composed of three gas cylinder models, a gas orifice model and a pressure regulating valve model. The main function is to provide the gas required for constant pressure operation to maintain working pressure for the rail-controlled engine; the propellant storage module consists of two storage tank models and liquid orifice models. The main effect is to store, distribute and supply the propellant required by the engine; the unit module mainly includes one orbit control engine model and sixteen attitude control thruster models, which provide propulsion for orbit control and attitude adjustment.

The power subsystem, as shown in Figure 4(c), includes two solar wing models, a battery model, a power controller model, and load models. The solar wing models on the left and right sides are composed of a certain number of solar cell models combined in series and parallel according to the design needs, and can generate full power under the condition of standard illumination of $1360W/m^2$. The battery model receives the electric energy output by the shunt regulator in the sunlit area for charging, and discharges when the solar wing power is insufficient for the load to maintain the stability of the bus voltage. The main function of the power controller model is to realize the distribution and regulation of solar wing power generation, battery charging and discharging, and load power, and to shunt according to needs. The load model is divided into two types, one is the fixed power load given in Figure 4(c), and the other is the load coupled in the other three subsystems. The second load is related to the working state of some equipment. For example, the power consumption of the electric load in the propulsion subsystem is related to the working status of seventeen engines.

The payload subsystem is generally composed of an-



(a) GNC subsystem model

(b) propulsion subsystem model

(c) power subsystem model

(d) payload subsystem model

Figure 4. Four main subsystem that make up satellite model in MWorks.

tenna models and transponder models, as shown in Figure 4(d). For the LEO communication satellite constellation, it is mainly used to establish ISL(L. Wang and D. Hu 2016; Yan 2010) and satellite-to-earth links. The user's communication data is received by the receiving antenna of the corresponding frequency band and then sent to the transponder. After the transponder performs gain adjustment and signal amplification processing on the data signal, it is transmitted to the transmitting antenna, and then to the next communication node through the transmitting antenna.

3.2.3 System Control Center Model

The system control center model belongs to the ground segment. The main function in the satellite constellation is to maintain the stability of the constellation structure. The input is orbital factors of all satellites calculated by the constellation orbit model, and the output is the speed pulse that the GNC subsystem needs to generate orbital maneuver control signal. In the satellite constellation structure, the two factors that have the greatest impact on the satellite coverage performance are the phase distribution and orbital plane distribution of the satellite, so phase control and orbital plane control are required.

The orbital plane and phase control reference of each satellite in the constellation are determined according to the following formula(Ulybyshev and Yuri 1998):

$$\Omega_0^* = \sum_{j=0}^{p-1}\sum_{k=0}^{s-1}\lambda_{jk}(\Omega_{jk} - j\cdot\frac{2\pi}{P} + j\cdot\frac{2\pi}{P}) \tag{7}$$

$$\begin{aligned} u_{00}^* = &\sum_{j=0}^{p-1}\sum_{k=0}^{s-1}\lambda_{jk}(u_{jk} - j\cdot F\cdot\frac{2\pi}{T} - k\cdot\frac{2\pi}{S}) \\ &+ j\cdot F\cdot\frac{2\pi}{T} + k\cdot F\cdot\frac{2\pi}{S} \end{aligned} \tag{8}$$

where λ_{jk} is the normalized weighting factor and the value is 1/T.

RAAN tolerance ε_Ω and phase tolerance ε_u are given in the form of model parameters. When $|\Delta u + H_u\Delta\dot{u}| > \varepsilon_u$, a phase holding maneuver is required, and the required tangential velocity pulse Δv_u is given by the following formula:

$$\Delta\dot{u}_r = \begin{cases} 0, & |\Delta u| < \varepsilon_u \\ K_u(\Delta u - \varepsilon_u), & \Delta u > \varepsilon_u \\ K_u(\Delta u + \varepsilon_u), & \Delta u < -\varepsilon_u \end{cases} \tag{9}$$

$$\Delta v_u = \frac{a}{3}(\Delta\dot{u} - \Delta\dot{u}_r) \tag{10}$$

where $\Delta\dot{u}_r$ is phase deviation change rate correction value. $\Delta\dot{u}$ is the rate of change of phase deviation, which is obtained by difference. $K_u = -1/H_u$, and H_u is the rate of change of phase deviation, which is obtained by difference.

Similarly, when $|\Delta\Omega + H_\Omega \Delta\dot{\Omega}| > \varepsilon_\Omega$, the orbital maintenance maneuver is required, and the required normal velocity pulse Δv_Ω is given by the following formula:

$$\Delta i = -\frac{1}{3J_2 \sin i}\left(\frac{a}{R_E}\right)^2\sqrt{\frac{a^3}{\mu}}\left(\Delta\dot{\Omega} - \Delta\dot{\Omega}_r\right) \qquad (11)$$

$$\Delta v_\Omega = 2\sqrt{\frac{\mu}{a}}\sin\left(i/2\right) \qquad (12)$$

where the calculation of $\Delta\dot{\Omega}_r$ is similar to that of $\Delta\dot{u}_r$. Δi is the required change in inclination. $J2$ refers to perturbation of the earth oblateness.

4 Simulation Results

In this section, some example parameters will be injected into system model. Among them, constellation orbit parameters are from Iridium NEXT(Iridium 2017), the specific content of constellation orbit parameters is described in section 3.2.1, which will not be repeated here; satellite parameters include satellite mass, engine thrust, solar cell rated voltage, and number of string cells or parallel cells, battery capacity and antenna gain, etc.; link parameters mainly include the longitude and latitude of the communication user, communication frequency, the minimum elevation angle of satellite-to-ground communication, and the minimum clearance height of ISL. The following article will analyze the system simulation results from three aspects: orbit coverage, communication link, and satellite status.

4.1 Orbit Coverage

Orbit coverage simulation mainly focuses on satellite position change, constellation structure maintenance and ground coverage. The position change of satellites is shown by the longitude and latitude of sub-satellite points. As shown in Figure 5, the longitude of the sub-satellite point of satellite 1-1 and satellite 1-2 is between $180°W$ and $180°E$, and the latitude of the sub-satellite point is between $85°S$–$85°N$. Figure 6 and Figure 7 show that under the action of the speed pulse signal generated by the system control center, orbit deviation can be controlled within a range of $\pm 2°$. Figure 8 shows that during the normal operation of the constellation, the two places A and B with coordinates $(160°E, 20°N)$ $(100°E, 40°N)$ have always been covered by satellites, and the maximum number can reach 5 satellites.

4.2 Communication Link

The focus of communication link simulation is to evaluate the communication quality of the two places and the impact of inter-satellite link switching on the communication quality. Figure 9 and Figure 10 show the link attenuation of the upper satellite-to-ground link, the lower satellite-to-ground link, and the inter-satellite link, respectively. Figure 11 shows the overall communication margin of the full

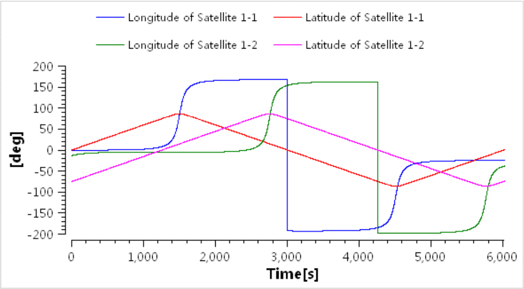


Figure 5. Longitude and latitude of sub-satellite points.

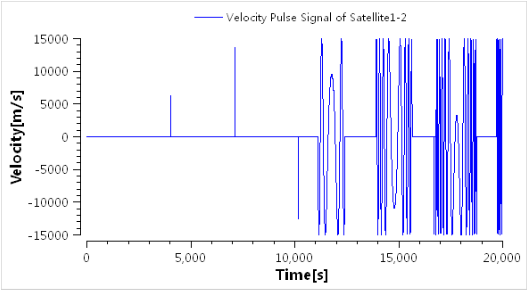


Figure 6. Tangential velocity pulse of satellite1-2.

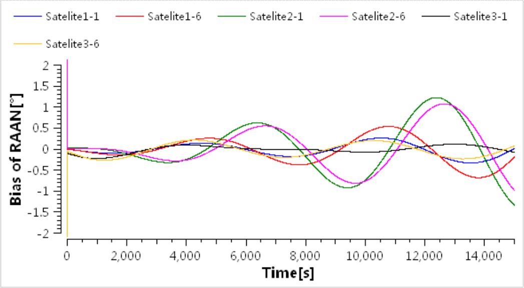


Figure 7. Deviation of orbital RAAN.

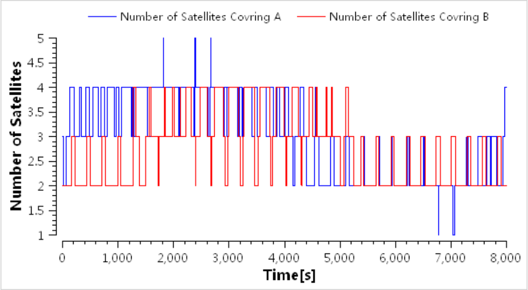


Figure 8. Number of satellites covered A and B.

link. Figure 12 shows the process of ISL switching. The simulation results show that the margin of the system can fully meet the communication needs of A and B.

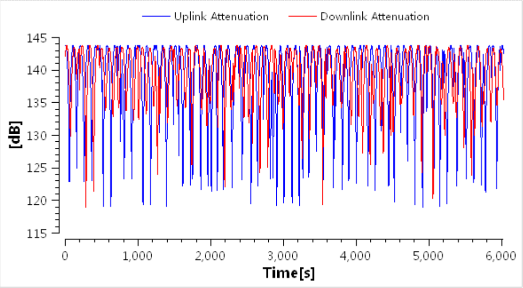


Figure 9. Satellite-to-groud link attenuation.

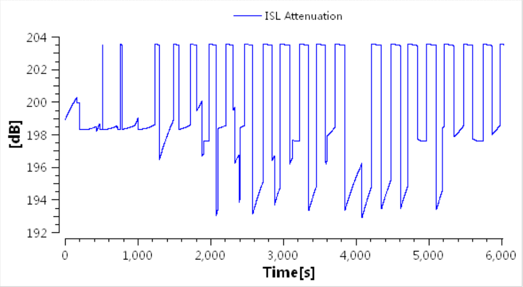


Figure 10. ISL link attenuation.

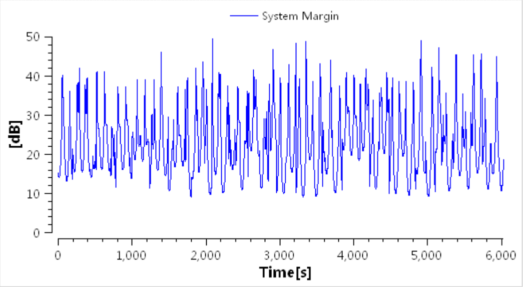
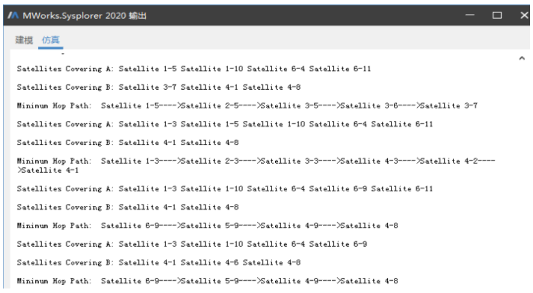


Figure 11. Link margin.



```
Satellites Covering A: Satellite 1-5 Satellite 1-10 Satellite 6-4 Satellite 6-11

Satellites Covering B: Satellite 3-7 Satellite 4-1 Satellite 4-8

Minimum Hop Path:  Satellite 1-5---->Satellite 2-5---->Satellite 3-5---->Satellite 3-6---->Satellite 3-7

Satellites Covering A: Satellite 1-3 Satellite 1-5 Satellite 1-10 Satellite 6-4 Satellite 6-11

Satellites Covering B: Satellite 4-1 Satellite 4-8

Minimum Hop Path:  Satellite 1-3---->Satellite 2-3---->Satellite 3-3---->Satellite 4-3---->Satellite 4-2----
>Satellite 4-1

Satellites Covering A: Satellite 1-3 Satellite 1-10 Satellite 6-4 Satellite 6-9 Satellite 6-11

Satellites Covering B: Satellite 4-1 Satellite 4-8

Minimum Hop Path:  Satellite 6-9---->Satellite 5-9---->Satellite 4-9---->Satellite 4-8

Satellites Covering A: Satellite 1-3 Satellite 1-10 Satellite 6-4 Satellite 6-9

Satellites Covering B: Satellite 4-1 Satellite 4-6 Satellite 4-8

Minimum Hop Path:  Satellite 6-9---->Satellite 5-9---->Satellite 4-9---->Satellite 4-8
```

Figure 12. ISL link swtiching.

4.3 Satellite Status

During the operation of the satellite constellation, the energy balance characteristics of 3 satellites in constellation are shown in Figure 13 and Figure 14. It can be seen that the state of charge of the three satellites remains above 80% and the output power of the solar array during the period with sunshine reaches 3000W, indicating the satellites energy system can satisfy the energy consumption under working conditions. Figure 15 shows the result of satellite attitude adjustment, the attitude angle is controlled within 1°.

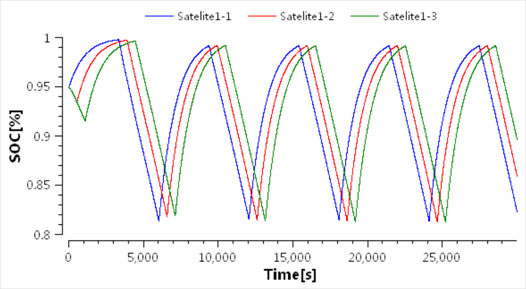


Figure 13. Results for SOC of batteries.

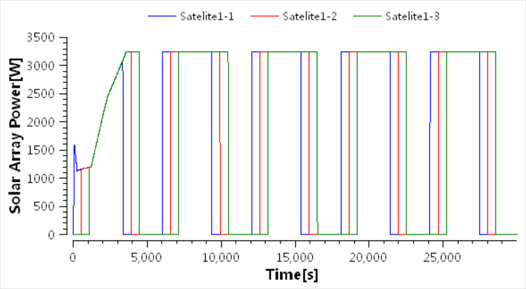


Figure 14. Results for output power of solar arrays.

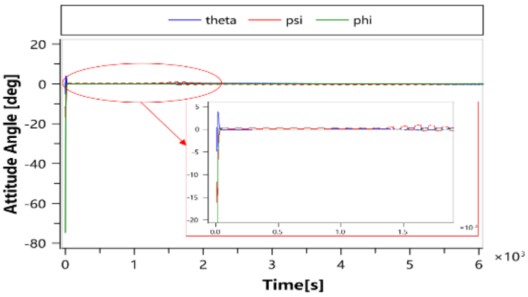


Figure 15. Results for satellite attitude angle.

5 Conclusions

In this paper, a system model of LEO communication satellite constellation is established, and a simulation analysis of typical service is carried out. The conclusions are as follows:

1. The system model established by the unified modeling language Modelica can support rapid design and verification of satellite constellation system.

2. The established system model can unify constellation orbit simulation, communication link performance simulation, and satellite status simulation.

3. The simulation application examples show that the constellation system model not only supports the characteristic analysis of key components, but also supports the overall performance evaluation of the system.

Acknowledgements

This work has been supported by National Key Research and Development Program of China(grant numbers 2019YFB1706501).

References

Chen, Changchun and Ying Lin (2020). “A novel design method for the constellation configuration stability considering the perturbation influence”. In: *Aerospace Shanghai* 01.37, pp. 33–37. DOI: 10.19328/j.cnki.1006-1630.2020.01.005.

Hu, Chenhua and Shengchun Wang (2018). “OPNET-based Simulation of LEO Satellite Communication System”. In: *Communications Technology* 51.10, pp. 2382–2388. DOI: 10.3969/j.issn.1002-0802.2018.10.018.

Iridium (2017-04-19). *Iridium NEXT is taking flight*. URL: https://www.iridium.com/network/iridiumnext.

Ulybyshev and Yuri (1998). “Long-Term Formation Keeping of Satellite Constellation Using Linear-Quadratic Controller”. In: *Journal of Guidance Control and Dynamics* 1.21, pp. 109–115. DOI: 10.2514/2.4204.

Wang, Liquan and Dongwei Hu (2016). “On-satellite routing and inter-satellite link design of low-orbit satellite mobile communication system”. In: *Information and Communication* 2, pp. 200–201. DOI: 10.3969/j.issn.1673-1131.2016.02.113.

Wang, Qian and Chao Xie (2021-04). “Simulation analysis of service performance of GPS M code satellite constellation based on STK software”. In: Navigation signal and signal processing. CNKI, pp. 134–137.

Yan, Jian (2010). “Research on the IP routing in LEO satellite constellation networks”. Doctoral dissertation. Tsinghua University. URL: https://kns.cnki.net/KCMS/detail/detail.aspx?dbname=CDFD0911&filename=1011280339.nh.

Zhang, Xiaodong (2008). “Research on Simulation and Modeling for Low Earth Orbit Constellation Satellite Communication System Based on OPNET”. Master’s thesis. National University of Defense Technology. URL: https://kns.cnki.net/kcms/detail/detail.aspx?FileName=2009213554.nh&DbName=CMFD2010.

Guidance, Navigation, and Control enabling Retrograde Landing of a First Stage Rocket

Christian Canham Meaghan Podlaski Luigi Vanfretti

Department of Electrical, Computer, and Systems Engineering
Rensselaer Polytechnic Institute
Troy, NY, United States
{canhac, podlam, vanfrl}@rpi.edu

Abstract

A Modelica model of a of a rocket's first stage is developed, designed to be representative of the launch vehicles in use in the United States in the late 2010s. The model uses initial conditions similar to those observed immediately after a second stage separation at 166 km altitude. A control system is developed enabling the rocket first stage to land back on Earth's surface at a predetermined landing pad in a controlled manner. The control system is evaluated based on its ability to compensate for altered initial conditions, as well as its ability to minimize acceleration forces and fuel consumption. The flight path of the simulated first stage rocket is compared to real-life telemetry data from a first stage rocket landing showing a similar trajectory.

Keywords: Rocket, Flight controller, GN&C, Retrograde Landing, Reaction Control Systems

1 Introduction

List of Acronyms and Definitions

Acronyms

GN&C · Guidance, Navigation, and Control
HIL · Hardware-in-the-Loop
IMU · Inertial Measurement Unit
LEO · Low Earth Orbit
NASA · National Aeronautics and Space Administration
PD · Proportional and Derivative Controller
RCS · Reaction Control System
STS · Space Transport System

Motivation

Rapid reusable space launch vehicles have long been a pursuit in the United States since it would dramatically increase accessibility to space. This was partially achieved with NASA's Space Transportation System (STS) Space Shuttle but failed to offer a fully reusable or low cost method. Rocket re-usability made significant strides when private space launch companies, including SpaceX and Blue Origin, demonstrated the ability to recover the first stage of the rocket by vertically landing it back on Earth's surface. This is achieved by relighting the rocket's engines in retrograde long enough to remove its horizontal and vertical velocities. This paired with gimbaled engines and control surfaces, such as grid fins, allow the first stage to be maneuvered back to a predetermined landing pad. The flight controller is responsible for making these engine and control surface control adjustments using input data from an inertial measurement unit (IMU) and GPS positional data.

Modeling and simulation of the launch vehicle is critical in the development in the GN&C control system. Hardware-in-the-loop (HIL) test beds are often times created to offer a low cost and rapid platform for the design of the control system and the calibration of their parameters before moving onto developmental prototypes.

The Modelica first stage rocket model and subsequent control system in this work represents early phase developmental activities that might occur when studying the feasibility of certain flight maneuvers. In this case, landing a rocket back on Earth after launching a payload into orbit. Many simplifications and assumptions are made in the first stage rocket model including the simplification of the rocket solely operating in the X-Z plane. Nevertheless, the control system core principles are fundamental and could be expanded to address these assumptions as the model grows in complexity. The key principle is to develop this control system despite these simplifications made in the model and show the ability to add features over time.

While Modelica models for a variety of aerospace applications have been successfully demonstrated (Wei et al. 2015; Re 2011; Briese, Klöckner, and Reiner 2017; Milz et al. 2019; Batteh et al. 2018; Posielek 2018; Hellerer, Bellmann, and Schlegel 2014), to the knowledge of the authors, there are no publicly available models similar to the one proposed in this work. The authors' believe that the growing interest and on-going advancements on rocket re-usability can benefit from the availability of an open source model that allows interested parties to exploit the advantages offered by object-oriented modeling provided by Modelica tools.

DOI
10.3384/ecp21181171

Contribution

This work is relevant to a user of the Modelica language looking for ways to rapidly develop a control system. In this case the control system is developed for a first stage rocket falling back to Earth but similar methods could be applied to other challenging control problems. The work shows how a simplified rocket model is nevertheless, an effective test-bed for the development of a control system that guides the rocket on a trajectory similar to that used by actual rockets, such as SpaceX's Falcon 9. The main contributions of the paper are the following:

- Implementation of the control system needed to recover a first stage rocket by vertically landing at a predetermined landing pad.
- Demonstration of Modelica's flexibility in creating models with increasing complexity leading to meaningful simulations.
- Documenting an open-source Modelica-based implementation of the aforementioned models available online at: `https://github.com/ALSETLab/RocketLanding`

Paper Organization

The paper is broken down in the following sections: Section 2 describes the first stage rocket model and the surrounding environment it operates in. Section 3 describes the control system designed to recover the first stage rocket by vertically landing at a predetermined landing pad. Section 4 compares the simulated Modelica rocket landing trajectory to real telemetry data from a SpaceX Falcon 9 rocket landing. Finally, Section 5 concludes the work.

2 Modelica First Stage Rocket Model

The model developed represents the first stage of a SpaceX Falcon 9 rocket immediately after second stage separation. It uses similar mass properties and initial conditions to those observed from publicly available SpaceX telemetry data (Pelham 2020). The model also includes the rocket engines and grid fins which are necessary to maneuver the rocket to the landing pad. Lastly the operating environment is taken into account by modeling drag on the vehicle as it falls through the Earth's atmosphere. The purpose was to design this model to be as representative of the actual rocket while making key simplifications that allow for the implementation of the control system presented in Section 3. The complete rocket model is shown in Figure 1.

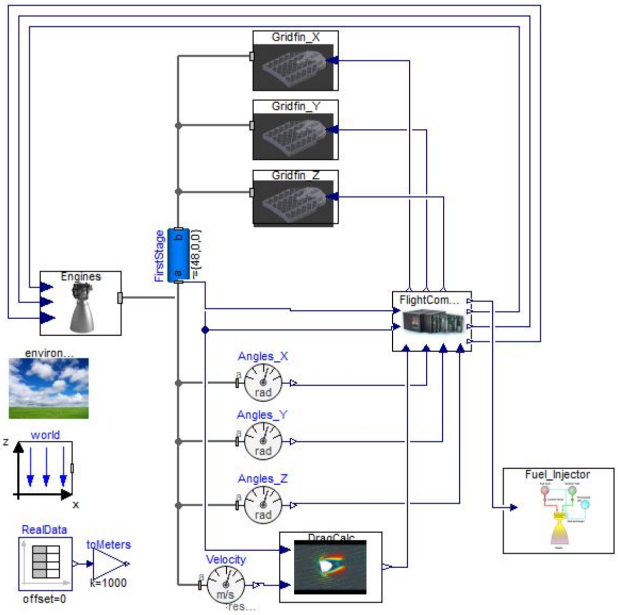


Figure 1. First stage rocket model

2.1 Mass Properties and Initial Conditions

The first stage rocket is modeled with a cylinder with a diameter of 3.7m and a length of 48 m. A density of 0.3g/cm^3 was selected based on the Falcon 9 rocket being composed of principally aluminum but with a mostly hollow interior.

In this model the frame of reference is the surface of the Earth which is simplified with a flat plane. The rocket is also assumed to be bounded to the X-Z plane. These two simplifications allow for easier control system development in Section 3 and more easily defined initial conditions. The initial conditions for the rocket are determined from live telemetry data from SpaceX's own webcasts (Pelham 2020). The simulation starts immediately after the second stage separation, where the first stage is assumed to be in orbit, and therefore, the vertical velocity in a flat Earth frame of reference is set to zero. The rocket fuselage also starts parallel to the Earth's surface. The initial velocity is purely measured by the horizontal component of 263 m/s in the X-axis. The altitude is set at 166km which is representative of the Falcon 9 stage separation for payload deployment in low earth orbit (LEO).

2.2 Engines and Grid Fins

The model includes a single rocket engine with a maximum thrust of 914 kN. This engine is simplified as a force acting in line with the fuselage of the rocket. The Falcon 9 rocket includes nine engines each with a maximum thrust of 914 kN but only one engine is typically used for landing.

The fuel consumption of the rocket engine is modeled in Figure 2 using fuel tanks and release valves that control the injection of liquid kerosene and oxygen into the engine at a proportion of 2:1 respectively. The same flight controller output that controls the simulated thrust from the engine is also used to control the fuel injector valves that release the fuel into the engine simulated with two tanks at ambient pressure. The flow volume through each valve is recorded and can be observed during the rocket simulation.

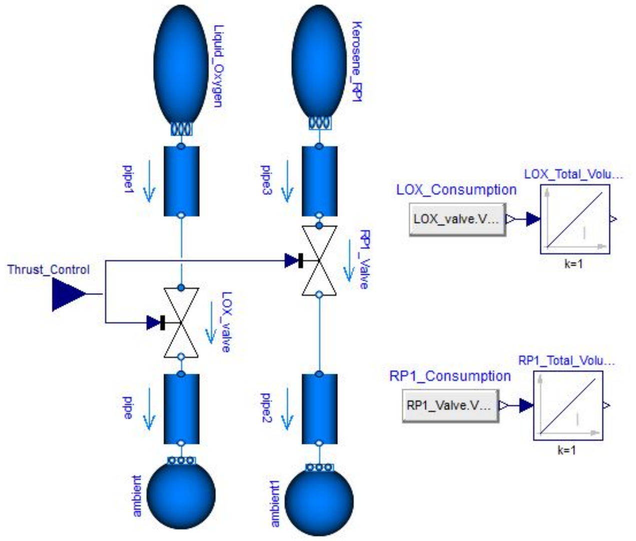


Figure 2. Fuel injector model

The model includes grid fins which impart a torque on the rocket depending on the grid fin's rotation angle relative to the fuselage. This allows the rocket to "steer" itself to a desired landing zone. On the Falcon 9 rocket these grid fins act as a control surface that redirect airflow therefore they are only effective at lower altitudes where the air is more dense. In the upper atmosphere Reaction Control Systems (RCS) are used to redirect the space vehicle using jets of compressed nitrogen. Both these actuators have the same intended effect so for simplification this model only uses the grid fins which are assumed to be equally effective at all altitude.

The torque on the rocket is a result of the rotating grid fins depending on the angle of the grid fin relative to the first stage fuselage. Maximum torque is imparted when the grid fin is at ±45° which decreases closer to zero degrees modeled as a sine function. At zero degrees the grid fin is perpendicular to the fuselage and no torque is applied.

The model includes three grid fins mounted 120° apart on the circumference of the rocket. Since the rocket is assumed to operate only in the X-Z plane only one grid fin is used to control the rockets rotation. The other two are simply used to keep the rocket from drifting out of the X-Z plane due to numerical artifacts in the simulation.

2.3 Reentry Drag

A space vehicle reentering the Earth's atmosphere experiences a considerable amount of drag and successive heating as a result of the high speeds acquired during orbital insertion - this velocity term is squared in the drag equation. If the reentry trajectory is optimized this inherit drag will help slow down the the vehicle reducing the need to use as much fuel with the rocket engines. For these reasons it is critical to model the atmospheric drag with accuracy.

The drag of the first stage rocket is modeled with a force vector normal to the direction of the rocket's motion. This drag force is calculated using equation 1 where the area A is calculated based on the rockets angle relative to the direction of motion. Velocity V accounts for all vertical and horizontal velocity components. The coefficient of drag was estimated based on the shape of the cylindrical aluminum fuselage. Selecting this coefficient of drag to be constant is a critical simplification. In reality this coefficient of drag would change based on the wake created behind the falling first stage. The air density is calculated using equation 2 where h is the altitude above sea level. Other variables in equation 2 are held constant and are based on the properties of air at sea level.

$$F_{Drag} = \frac{1}{2} C_D V^2 A \rho \tag{1}$$

$$\rho = \rho_0 \left(1 - \frac{Lh}{T_0}\right)^{\frac{gM}{RL}} \tag{2}$$

3 Modelica Flight Controller

After developing the rocket model, the next step was to create the GN&C flight controller that guides the first stage down to the landing pad. A diagram of the full flight controller is pictured in Figure 3.

The flight controller model is further broken down into four different controllers. The first is the grid fin controller in Section 3.1 which is directly responsible for controlling the angle of each of the three grid fins. The other three controller are related to the engine thrust. Each of these three controllers take control of the vehicle at different phases of flight but their combined outputs are summed so if need be they could be working together simultaneously. These three controller include the boost-back controller in Section 3.2, the re-entry controller in Section 3.3, and the landing controller in Section 3.4.

The flight controller takes as input the altitude and horizontal X-axis displacement relative to the landing pad. On an actual rocket these parameters would be obtained though pressure or GPS signals.The flight control also has inputs for the angle from all three axes. Normally these would be obtained through an internal IMU internal to the flight controller. The flight controller also has an internal clock for landing sequences that require a set duration.

As outputs the flight controller has angles for each of the three grid fins. The flight controller also has three outputs for each of the three axes components in the engine thrust. The engine model then uses these thrust components to determine the total thrust force vector.

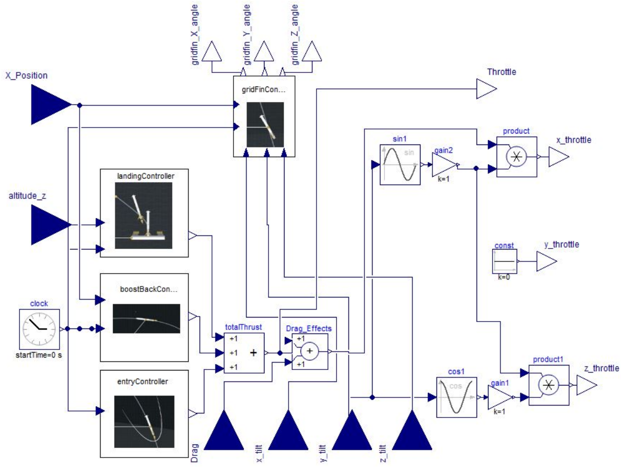


Figure 3. Flight controller model

3.1 Grid Fin Controller

The grid fin controller in Figure 4 is responsible for determining the angle of all three grid fins which impart a torque on the rocket allowing it to maneuver to the landing pad. As mentioned in Section 2.2 the rocket is assumed to operate only in the X-Z plane. Therefore, only the Y-axis grid fin is responsible for guiding the rocket to the landing pad. The other two grid fins are simply present to keep the first stage rocket from drifting out the X-Z plane. Numerical artifacts from the simulation solver have shown to result in slight drifting which then compounds until the rocket is out of control and tumbling through the atmosphere.

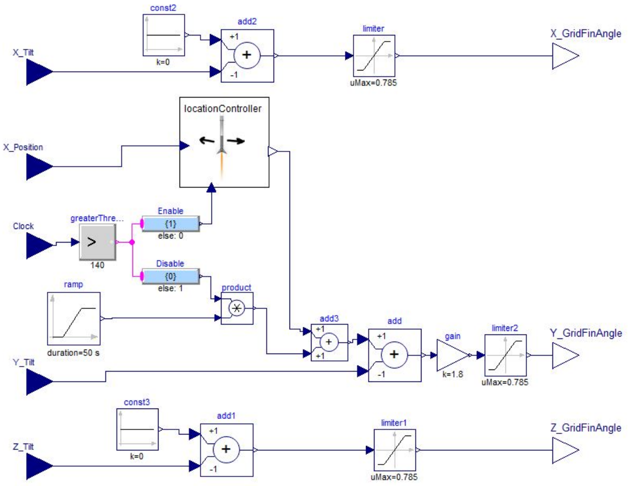


Figure 4. Grid fin controller model

The grid fin model discussed in Section 2.2 applies a maximum torque when rotated to 45 degrees relative to the first stage fuselage length. For this reason the grid fin controller applies a hard limit at $\pm\pi/4$ radians for each of the grid fin input angles. Angles in excess of 45° decrease the applied torque. As such there is no reason to operate in those regions.

The Y-axis grid fin is critical in accomplishing two tasks. The first is maneuvering the first stage rocket to an upright vertical position relative to the Earth's surface. As discussed in Section 2.1 the first stage initially only has a horizontal velocity. Once the rocket engines nearly remove this horizontal velocity the rocket needs to be rotated to a vertical position so the engines can begin slowing down the rocket in the Z-axis. This rotation is accomplished by comparing the rocket's rotational angle with that of a gradual ramp function which initiates at a predetermined altitude. A simple proportional controller relative to the changing ramp function is used to bring the rocket to the upright position. Even after the ramp function has finished bringing the rocket to the vertical position it still plays a critical roll in keeping the rocket vertical as it comes down for a landing.

The second task of the Y-axis grid fin is to guide the rocket first stage to the landing pad. This is handled by the location controller which operates within the grid fin controller. This location controller is only enabled after the rocket has transitioned to the upright position. The location controller uses proportional and derivative gains using the horizontal displacement from the landing pad as input. The gains within this PD controller were later tweaked upon evaluated the flight trajectory of the rocket.

3.2 Boost-Back Controller

The initial 263 m/s horizontal X-axis velocity gained during orbital insertion needs to be removed in order to land the first stage back at the landing pad. This is achieved by firing the rocket engines in retrograde where the thrust of the engine is normal to the direction of travel. The rocket maintains its horizontal angle during this flight phase with the help of the grid fin controller discussed in Section 3.1.

As input, the boost-back controller takes the horizontal displacement relative to the landing pad as well and an internal clock. The controller directs the engine to fire at time zero and continue until the horizontal velocity falls below a certain thresh-hold value of 50 m/s. Some horizontal velocity is desirable in order to minimize the flight path distance taken by the first stage rocket.

3.3 Reentry Controller

Space vehicles experience considerable heating during reentry into the Earth's atmosphere. The space shuttle and smaller bluff body space capsules use heat shielded tiles to protect the spacecraft and the payload inside. A first stage rocket reentering the Earth's atmosphere doesn't have this luxury of a heat shielded exterior since critical external components like the engines will be exposed regardless. To mitigate this excessive heating a rocket can instead use its own engines to slow down its velocity.

The purpose of the reentry controller is to conduct a 30 second engine burn during the critical phase of flight where aerodynamic forces are the greatest. This reentry burn decreases the rocket velocity considerably minimizing excessive heating. During this entry burn the grid fin controller discussed in Section 3.1 keeps the rocket pointed normal to the direction of motion thereby maximizing the effectiveness of the entry burn in slowing down the rocket.

3.4 Landing Controller

The final controller is responsible for landing the rocket first stage in a controlled manner that minimizes excessive accelerations. This landing controller takes the altitude and an internal clock signal as input.

Once the rocket's altitude falls below the threshold altitude of 5 km, a PD controller within the landing controller is enabled which takes over in guiding the rocket safely down to the landing pad. The gain values in this PD controller were determined through simulation. Ideally the vertical velocity component of the rocket should be zero at the moment of touch down. Simultaneously, the rocket should not undershoot the ground else it would end up hovering thereby wasting fuel. A hovering rocket is also no longer easily maneuverable since the grid fins are only effective when a velocity component is present.

4 Model Validation

The completed model and early iteration flight controller were simulated with performance evaluated based on the rocket's ability to land in a 50 m diameter landing pad. These early simulations were critical in the identification of PD controller gain constants used during different phases of flight. Landing controller PD constants were selected such that the touch down velocity was less than 2 m/s. Meanwhile, grid fin controller PD gains were selected to guide the rocket to the landing pad while minimizing overshoot and horizontal oscillations.

With all flight controllers tuned the rocket first stage was successfully able to land at the predetermined landing pad. Figure 5 shows three different flight paths of the rocket first stage with altitude on the y-axis and horizontal displacement plotted on the x-axis. Each of the four curves shown use different initial conditions for the horizontal velocity. The curves in green show successful landings where the first stage was able to touch down within the 50 m diameter predetermined landing pad. The initial condition originally selected for this model was was 293 m/s. The success of these other two trajectories show the flight controller is robust against initial conditions of at least ±10 m/s. The curve in red shows an unsuccessful landing where the rocket undershoot the pad. In this case an initial condition of 240 m/s was not enough horizontal velocity to carry the rocket close to the landing pad before entering the Earth's atmosphere where its horizontal velocity is nearly removed.

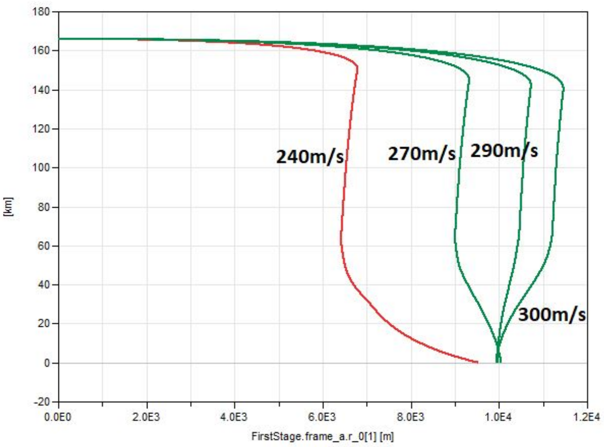


Figure 5. Flight trajectories

This first stage rocket model was designed to be representative of the SpaceX Falcon 9 in terms of mass properties, initial conditions, and actuator capabilities. A critical next step was to validate this simulated Falcon 9 rocket first stage landing against actual SpaceX telemetry data (Pelham 2020). Figure 6 shows the simulation trajectory in blue and the SpaceX Falcon 9 telemetry data in red. Altitude is plotted on the Y-axis with time one the x-axis. Comparison of the two trajectories show near identical overlap. Overlaid onto the plot are four boxes showing different phases of flight where key maneuvers occur. Each of these maneuvers is controlled by one of the controllers discussed in Section 3. One subtle difference between the simulation and SpaceX trajectories can be observed in the the landing zone box from 180 - 250 sec. The simulation data shows the first stage rocket slowing down earlier resulting in the rocket maintaining a higher altitude. The SpaceX Falcon 9 meanwhile appears to wait for a lower altitude before relighting its engines resulting in higher accelerations. This lower engine burn is likely attributed to the desire to minimize fuel on landings in order to maximize payload carrying capabilities during launch.

To better visualize the trajectory of the first stage rocket, a virtual simulation environment was created using the DLR Visualization library (Hellerer, Bellmann, and Schlegel 2014) shown in Figure 7. Using this virtual environment, a user can track the rocket as it maneuvers through the different phases of flight including visualization of grid fin rotation, engine burns, and landing at the pad at Cape Canaveral, FL.

5 Conclusions

The rocket model developed in this work and subsequent flight controller demonstrates the basic control fundamentals in recovering a rocket by landing vertically back on Earth. In this case, the rocket model was based on the SpaceX Falcon 9 first stage which has proven itself in its ability to land in a controlled manner allowing for multiple reuses. Comparing the simulated flight trajectory to data from a Falcon 9 landing shows nearly identical flight characteristics. While normally these types of simulations would occur in the early development iterations of a control system rather than recreating one *a posteriori*, there is value in having a open source Modelica model of a rocket landing. Even with the primary goal of recovery demonstrated in simulation and if real-life, there is likely opportunity for optimization of these flight paths with regards to minimizing fuel consumption and accelerations.

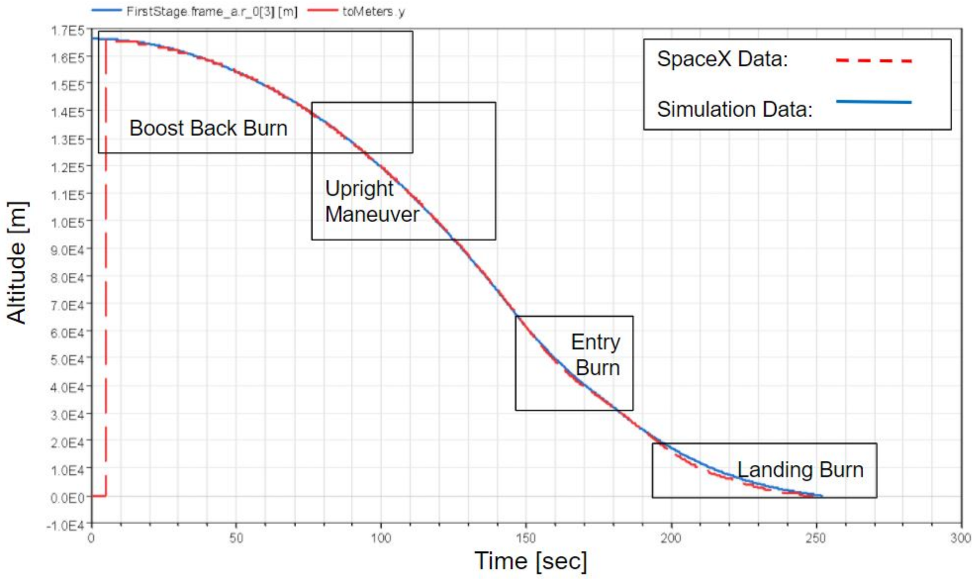


Figure 6. Comparison to Falcon 9 data

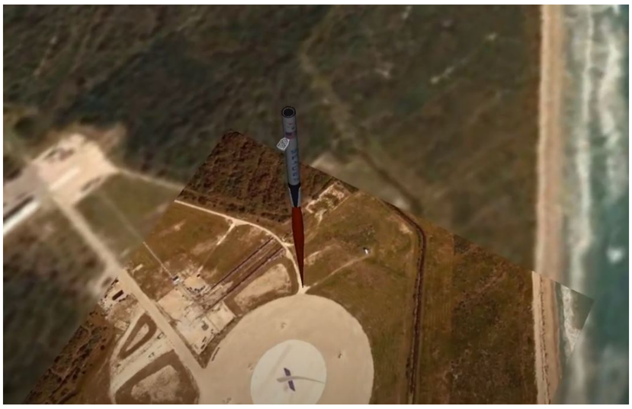


Figure 7. Visualization of rocket landing

Acknowledgements

This work was supported in part by the National Aeronautics and Space Administration through the University Leadership Initiative Award Number 80NSSC19M0125 for the Center for High-Efficiency Electrical Technologies for Aircraft (CHEETA). The second author is supported through the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE 1744655 and the Chateaubriand Fellowship of the Office for Science & Technology of the Embassy of France in the United States.

References

Batteh, John et al. (2018-11). “Development and Implementation of a Flexible Model Architecture for Hybrid-Electric Aircraft”. In: *Proceedings of the 1st American Modelica Conference, Cambridge, MA, USA*, pp. 37–45. DOI: 10.3384/ecp1815437. URL: https://ep.liu.se/konferensartikel.aspx?series=ecp&issue=154&Article_No=4.

Briese, Lâle Evrim, Andreas Klöckner, and Matthias Reiner (2017-05). “The DLR Environment Library for Multi-Disciplinary Aerospace Applications”. In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic*, pp. 929–938. DOI: 10.3384/ecp17132929. URL: https://ep.liu.se/ecp/article.asp?issue=132&article=102&volume=0#.

Hellerer, Matthias, Tobias Bellmann, and Florian Schlegel (2014-03). “The DLR Visualization Library — Recent Developments and Applications”. In: *Proceedings of the 10th International Modelica Conference, Lund, Sweden*, pp. 899–911. DOI: 10.3384/ECP14096899. URL: https://ep.liu.se/en/conference-article.aspx?series=ecp&issue=96&Article_No=94.

Milz, Daniel et al. (2019-03). “Advances in Flight Dynamics Modeling and Flight Control Design by Using the DLR Flight Visualization and Flight Instruments Libraries”. In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany*, pp. 481–488. DOI: 10.3384/ecp19157481. URL: https://ep.liu.se/ecp/article.asp?issue=157&article=049&volume=0.

Pelham, Jonathan (2020-05). “SpaceXtract”. In: *Extraction and analysis of telemetry from rocket launch webcasts*. URL: https://github.com/shahar603/SpaceXtract.

Posielek, Tobias (2018-11). “A Modelica Library for Spacecraft Thermal Analysis”. In: *Proceedings of the 1st American Modelica Conference, Cambridge, MA, USA*, pp. 46–55. DOI: 10.3384/ecp1815446. URL: https://ep.liu.se/konferensartikel.aspx?series=ecp&issue=154&Article_No=5.

Re, Fabrizio (2011). “Modelica Landing Gear Modelling and On-Ground Trajectory Tracking with Sliding Mode Control”. In: *Advances in Aerospace Guidance, Navigation and Control*. Ed. by Florian Holzapfel and Stephan Theil. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 103–115. ISBN: 978-3-642-19817-5.

Wei, Liu et al. (2015-09). “Modeling and Simulation of Liquid Propellant Rocket Engine Transient Performance Using Modelica ”. In: *Proceedings of the 11th International Modelica Conference, Versailles, France*, pp. 485–490. DOI: 10.3384/ecp15118485. URL: https://ep.liu.se/en/conference-article.aspx?series=118&issue=118&Article_No=52.

An Ice Storage Tank Modelica Model: Implementation and Validation

Guowen Li[1] Yangyang Fu[1] Amanda Pertzborn[2] Jin Wen[3] Zheng O'Neill[1]

[1] J. Mike Walker '66 Department of Mechanical Engineering, Texas A&M University, College Station, TX
{guowenli, yangyang.fu, zoneill}@tamu.edu

[2]National Institute of Standards and Technology, Gaithersburg, MD
{amanda.pertzborn@nist.gov}

[3]Department of Civil, Architectural and Environmental Engineering, Drexel University, Philadelphia, PA
{jinwen@drexel.edu}

Abstract

Energy storage systems have been gaining attention as a means of load management in grid-interactive efficient buildings. This study investigated the physics of the ice storage tank (IST) and implemented an IST model in Modelica. The developed IST Modelica model was compared with a similar model in EnergyPlus and was validated against experimental data from a testbed at the National Institute of Standards and Technology. Three statistical performance metrics were used to quantify the accuracy of the IST model. Validation results (CV(RMSE) ≤ 10.20 %, NMBE ≤ 0.44 %) show that the proposed model has a good prediction accuracy according to ASHRAE Guideline 14.

Keywords: Ice storage tank, Modelica modeling, Model validation

1 Introduction

Ice thermal storage systems have been proven to be effective in reducing the cost of energy for operating buildings by shifting cooling demand from on-peak periods with high electricity prices to off-peak periods with lower electricity prices. The ice storage tank (IST) is a key component in an ice thermal storage system. By applying proper control strategies to the IST, research studies have shown significant cost savings including energy costs and demand reduction costs (e.g., 10 % to 55 %), which makes IST an attractive financial option for buildings (Braun 1990; Henze 2003; Candanedo 2013). Control-oriented modeling and simulation is an effective way to evaluate the system performance under different control strategies. Jekel (1991) created a model of a static ice-on-coil IST based on basic heat transfer relationships and analysis; the author used TRNSYS (Beckman 1994) to model a variable flow air-conditioner system connected with the IST. Both the charging and discharging periods of the tank operation were modeled and compared with manufacturers' data (Calmac ice tank model 1190 with working fluid of 25 % ethylene glycol). The prediction errors for the charging and discharging period were within 12 % and 10 % of the manufacturer's performance data, respectively. Ihm et al. (2004) developed an ice-based thermal storage model for EnergyPlus. This IST model uses the building load and system thermodynamic models for two direct ice systems (i.e. ice-on-coil external melt and ice harvester) and one indirect ice system (i.e. ice-on-coil internal melt). For the external (or internal) melt thermal storage tank, the brine flowing through coils charges and discharges the tank on the outside (or inside) of the coils. The thermal storage model systems were integrated as part of the EnergyPlus cooling plant components. Candanedo et al. (2013) numerically investigated the impact of different control strategies for a simplified ice storage system developed in EnergyPlus.

In terms of building energy and control system modeling, traditional building energy simulators, including TRNSYS and EnergyPlus, have the following limitations. First, these traditional simulation programs intertwine model equations and numerical solvers in their source code. The lack of separation between model equations, data, and solvers makes it hard for their models to support some use cases, especially when different control strategies and designs are involved (Wetter 2009). Second, some platforms, which are inherently designed for a steady-state simulation, are not suitable for evaluating the system dynamics, and the semantics of their control modules have little in common with how actual control works (Fu 2019). For example, EnergyPlus does not model local controllers (e.g., proportional-integral (PI) controller) for a building energy system. Additionally, the dead band and waiting time commonly used in building controls are not considered in EnergyPlus. The idealization of control makes it difficult to investigate, implement and verify actual control strategies in simulation (Wetter 2011; Fu 2018).

Certain commercial equipment, instruments, or materials are identified in this paper in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

DOI
10.3384/ecp21181177

To address these problems, the equation-based modeling language Modelica (Mattsson 1998) has been utilized to model and simulate building energy and control systems. The open-source Modelica Buildings Library (MBL) was developed by Lawrence Berkeley National Laboratory for typical building energy and control system modeling and simulation (Wetter 2014). However, the latest release of the MBL does not support the modeling of ice storage tank systems due to the lack of ice tank models.

This study implemented and validated an IST Modelica model based on MBL to support control-oriented studies, which could extend MBL to enable ice storage systems modeling and simulation. The rest of this paper is organized as follows: Section 2 describes the mathematical equations of the IST model and three statistical metrics to quantify and evaluate the model accuracy. Section 3 discusses how the IST model was implemented in the Modelica environment using the MBL. Section 4 validates the implemented IST model in Modelica against a similar model in EnergyPlus and experimental data from the National Institute of Standards and Technology (NIST). The final section is the conclusion.

2 Methodology

2.1 IST Mathematical Model

The mathematical model of the IST is based on the EnergyPlus model (Strand 1992) presented in Eqs. (1) to (8). The detailed IST model allows the users to model more closely specific manufacturers' ice storage units due to the use of curve fits. In section 4, the IST model was validated against an actual ice tank at NIST which is the type of ice-on-coil internal melt. The IST has three modes: a dormant mode when the storage is not engaged in operation, a discharging mode when the storage discharges cooling energy to the warm brine, and a charging mode when the storage is charged with the cold brine. When the tank is dormant, there is no fluid passing through the tank and the outlet temperature is considered to be equal to the inlet temperature. Details about the discharging mode and charging mode are shown.

- Discharging Mode

$$\begin{aligned}\dot{q}^* \times \Delta t = \\ d_1 + d_2(1-P_{ch}) + d_3(1-P_{ch})^2 + \\ [d_4 + d_5(1-P_{ch}) + d_6(1-P_{ch})^2]\Delta T_{lm}^*\end{aligned} \tag{1}$$

$$\dot{q}^* = \frac{\dot{q}}{Q_{sto}} \tag{2}$$

$$\dot{q} = \dot{m}c_p(T_{out} - T_{in}) \tag{3}$$

$$\Delta T_{lm}^* = \frac{\Delta T_{lm}}{\Delta T_{nom}} \tag{4}$$

$$\Delta T_{lm} = \frac{\Delta T_1 - \Delta T_2}{\ln\left(\frac{\Delta T_1}{\Delta T_2}\right)} = \frac{T_{in} - T_{out}}{ln\left(\frac{T_{in} - T_{fre}}{T_{out} - T_{fre}}\right)} \tag{5}$$

where, $\dot{q}^*$ is the normalized instantaneous heat transfer rate between the brine and the ice in the tank; Δt is the time step of the operation data used in the curve fitting for discharging coefficients d_1 to d_6 ; P_{ch} is the fraction charged; ΔT_{lm} is the logarithmic mean temperature difference (LMTD); ΔT_{lm}^* is the LMTD between the inlet and outlet temperature of the tank normalized by a nominal temperature difference, ΔT_{nom}. Physically, $\dot{q}^*$ is defined as the ratio of the instantaneous heat transfer rate $\dot{q}$ to the total latent storage capacity Q_{sto}; $\dot{q}$ is negative when the tank is discharged. T_{in} is the tank inlet temperature, T_{out} is the tank outlet temperature, and T_{fre} is the freezing temperature of water or the latent energy storage material.

Eq. (5) is not numerically robust due to singularities that occur in the following scenarios: 1) $\Delta T_1 = \Delta T_2$, which causes a denominator of zero; 2) $\Delta T_1 = 0$ or $\Delta T_2 = 0$, which violates the logarithm function; 3) ΔT_1 and ΔT_2 have different signs. For a robust implementation of the LMTD calculation in the Modelica numerical environment, Eq. (5) is smoothed over different regions as shown in Eq. (5.1) to Eq. (5.5). The function $f_{smoothMax}(y, y_{lim})$ provides a continuously differentiable approximation for the variable y, which can be no less than the limiting value y_{lim}. The function $f_{smoothMin}(y, y_{lim})$ limits the variable y to be no larger than y_{lim}, where $\Delta y = y - y_{lim}$, and δ is used for regularization. When $|\Delta y| < \delta$, a second order polynomial function is used to create a smooth transition from y to y_{lim}. The smoothed functions not only help avoid the occurrence of zero in the denominator, but also ensure the continuity and differentiability of the simulated data.

$$\Delta T_1 \ = f_{smoothMax}(T_{in} - T_{fre}, T_{lim}) \tag{5.1}$$

$$\Delta T_2 \ = f_{smoothMax}(T_{out} - T_{fre}, T_{lim}) \tag{5.2}$$

$$\Delta T_{lm} = f_{smoothMin}\left(\frac{\Delta T_1 - \Delta T_2}{\ln\left(\frac{\Delta T_1}{\Delta T_2}\right)}, \Delta T_{lim}\right) \tag{5.3}$$

$$f_{smoothMax}(y, y_{lim}) = \begin{cases} y, & (y > y_{lim} + \delta) \\ y_{lim}, & (y < y_{lim} - \delta) \\ \frac{y + y_{lim}}{2}, & (|\Delta y| = \delta) \\ \frac{\frac{\Delta y^2}{\delta}\left(3 - \frac{\Delta y^2}{\delta^2}\right)}{4} + \frac{y + y_{lim}}{2}, & (|\Delta y| < \delta) \end{cases} \tag{5.4}$$

$$f_{smoothMin}(y, y_{lim}) = \begin{cases} y_{lim}, & (y > y_{lim} + \delta) \\ y, & (y < y_{lim} - \delta) \\ \frac{y + y_{lim}}{2}, & (|\Delta y| = \delta) \\ \frac{\frac{\Delta y^2}{\delta}\left(\frac{\Delta y^2}{\delta^2} - 3\right)}{4} + \frac{y + y_{lim}}{2}, & (|\Delta y| < \delta) \end{cases} \tag{5.5}$$

- Charging Mode

When the thermal tank is being charged, the charging heat transfer is calculated by Eq. (6). The constant parameters c_1~c_6 are charging coefficients.

$$\dot{q}^* \times \Delta t = c_1 + c_2 P_{ch} + c_3 {P_{ch}}^2 + [c_4 + c_5 P_{ch} + c_6 {P_{ch}}^2] \Delta T_{lm}^* \qquad (6)$$

For both discharging mode and charging mode, the mass of ice in the tank is calculated by Eqs. (7) and (8), where SOC is the state of charge that indicates the mass ratio (0 %-100 %) of ice in the tank, SOC' is the derivative of SOC, H_f is the latent heat of fusion for water at 0 °C, m_{ice} is the mass of ice in the tank, $m_{ice,max}$ is the maximum ice capacity of the tank, and $\dot{m}$ is the mass flow rate of the fluid.

$$SOC' = \frac{\dot{m} c_p (T_{out} - T_{in})}{m_{ice,max} H_f} \qquad (7)$$

$$SOC = \frac{m_{ice}}{m_{ice,max}} \qquad (8)$$

2.2 Validation Metrics

To evaluate the accuracy of the proposed model, three statistical metrics are applied: Coefficient of Variation of Root Mean Square Error (CV(RMSE)), the coefficient of determination (R^2), and the Normalized Mean Bias Error (NMBE). These metrics are defined in Eq. (9) to Eq. (12).

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2}{n}}, \qquad (9)$$

$$CV(RMSE) = \frac{RMSE}{\bar{Y}_i}, \qquad (10)$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2}{\sum_{i=1}^{n} (Y_i - \bar{Y}_i)^2}, \qquad (11)$$

$$NMBE = \frac{\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2}{(n-p) \times \bar{Y}_i}, \qquad (12)$$

where, Y_i is the measured data, $\hat{Y}_i$ is the predicted data, n is the number of data points, $\bar{Y}_i$ is the mean value of the measured data, p is the number of parameters in the numerical model.

According to ASHRAE Guideline 14 (ASHRAE 2014), the predicted model shall have an NMBE up to 5 % and a CV(RMSE) up to 15 % using monthly calibration data. If hourly calibration data are used, these requirements shall be 10 % and 30 %, respectively.

3 Modelica Modeling

3.1 Model Description

The IST Modelica model contains four key components as presented in Figure 1: LMTD calculator, heat flow rate calculator, storage mode selector, and outlet temperature controller. The storage mode selector sends the mode signal (discharging, charging, dormant) to the LMTD calculator and the outlet temperature controller that controls the outlet temperature to the setpoint. The heat flow rate calculator outputs the SOC and ice mass. Table 1 summarizes the key components in the IST Modelica model.

Table 1. Description of the IST Modelica model.

Components	*Model Description*
LMTD Calculator	LMTD algorithm with *smooth* functions
Heat Flow Rate Calculator	Polynomial coefficients of curve fitting data
Storage Mode selector	Discharging, charging, dormant mode
Outlet Temp Controller	PI control for the main valve and bypass valve

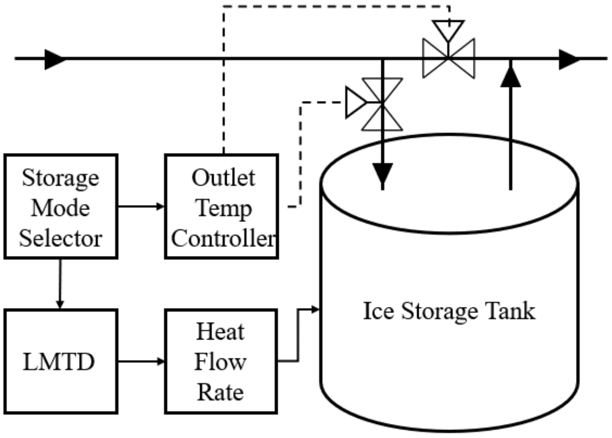


Figure 1. Schematic diagram of the IST model.

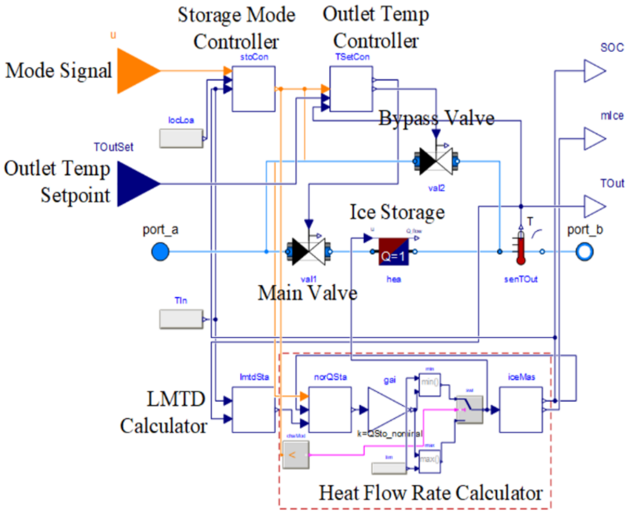


Figure 2. Modelica model of the ice storage tank.

3.2 Model Components

Figure 2 presents the detailed components inside the IST Modelica model; details of each component are described below.

- LMTD Calculator

LMTD is the key intermediate variable that determines the calculation process of the whole model, which is calculated by Eqs. (5.1) - (5.5).

- Heat Flow Rate Calculator

The heat flow rate is calculated by Eq. (1) and Eq. (6) for discharging and charging mode, respectively. When the tank is dormant, the heat flow rate is assumed to be zero.

- Storage Mode Selector

The storage mode selector determines the operating mode of the tank (i.e., dormant, discharging, or charging modes) in response to measured system states such as SOC, the flow rate, and the inlet temperature of the coolant.

The state diagram of the storage mode selector is shown in Figure 3. If the mass flow rate is greater than the minimum flow rate, the inlet temperature is greater than the freezing temperature of water plus a temperature tolerance ($dT_{if,min}$), and SOC is greater than a discharging tolerance, then the ice storage tank is in the discharging mode. If the mass flow rate is greater than the minimum flow rate, the inlet temperature is less than the freezing temperature of water minus a temperature tolerance, and SOC is less than a charging tolerance, then the ice storage tank is in the charging mode. Otherwise, the tank is dormant and bypassed. Figure 4 shows the state graph diagram implemented in Modelica, which has four input signals and one output signal, the storage mode.

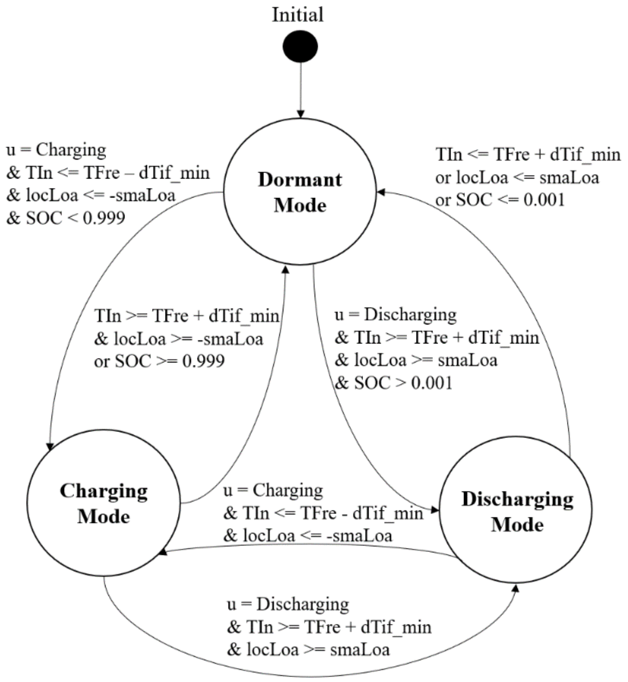


Figure 3. State diagram of the storage mode selector.

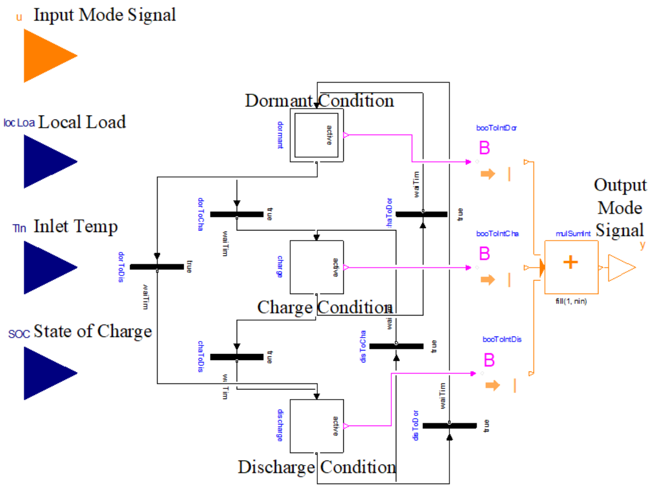


Figure 4. Modelica diagram of the storage mode selector.

- Outlet Temperature Controller

The IST outlet temperature is maintained at its setpoint by adjusting the bypass valve position through a built-in PI controller. The control values and diagram of the outlet temperature controller are shown in Table 2 and Figure 5, where K1 is the opening value of the main valve, K2 is the opening value of the bypass valve, and u_{PI} is the output value of a built-in PI controller. If the tank is dormant, the main valve will be closed (K1=0) and the bypass valve will be fully open (K2=1). If the tank is charged, the main valve will be fully open (K1=1) and the bypass valve will be closed (K2=0). If the tank is discharged, the PI controller will adjust the main valve and the bypass valve to meet the outlet temperature setpoint (K1=1- u_{PI} , K2= u_{PI}).

Table 2. Control values of the outlet temperature controller.

Mode	Dormant	Charging	Discharging
Main Valve	Off (K1 = 0)	On (K1 = 1)	On (K1 = 1-u_{PI})
Bypass Valve	On (K2 = 1)	Off (K2 = 0)	On (K2 = u_{PI})

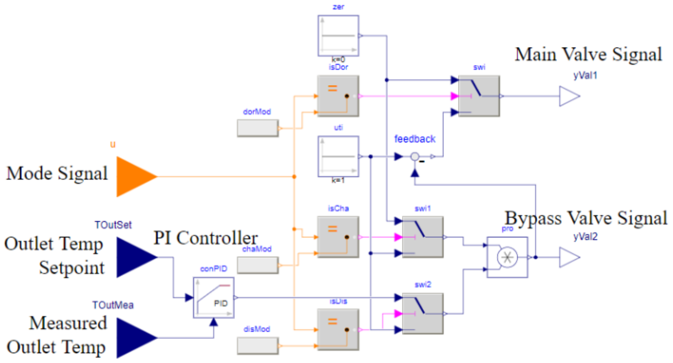


Figure 5. Modelica diagram of the outlet temperature controller.

4 Comparison and Validation

This section presents the comparison and validation of the IST Modelica model against two data sources: 1) a similar model in EnergyPlus and 2) measurement data from NIST. Three accuracy metrics are presented.

4.1 Modelica Model vs. EnergyPlus Model

We selected a built-in IST model in an EnergyPlus ice tank example file that is based on the same mathematical model (Strand 1992) presented in Eqs. (1) to (8). Table 3 lists the details of the EnergyPlus model with modified parameters (cooling capacity, polynomial coefficients of charging curve data, and medium type). The IST EnergyPlus model was simulated for about 8 hours (from 10 a.m. to 6 p.m.) of discharging operation and 2 hours (from 0 a.m. to 2 a.m.) of charging operation on July 21st using typical meteorological year (TMY3) weather data. Then the simulated dataset (inlet/outlet temperature of IST, mass flow rate of chilled water, and SOC) from EnergyPlus was exported for use in the Modelica virtual testcase.

Table 3. Description of the IST EnergyPlus model.

Descriptions	*EnergyPlus model*
Filename	5ZoneDetailedIceStorage. idf
Weather Data	Chicago-Midway AP 725340 (TMY3)
Floor Area	463.6 m^2
Ice Storage Capacity	0.05 GJ
Number of Story	1
Number of Zones	6
Timestep of Simulation	1 min
Discharging Curve	$d = [0.0, 0.09, -0.15,$ 0.612,-0.324,-0.216]
Discharging Time	10 a.m. to 6 p.m., July 21st
Charging Curve	$c = [0.318, 0, 0, 0, 0, 0]$
Charging Time	12a.m. to 2a.m., July 21st
Medium	30 %PG (propylene glycol) + 70 %Water

A virtual testcase was built for the IST Modelica model as presented in Figure 6. The Modelica virtual testcase uses the same IST parameters and inlet conditions (mass flow rate, temperature, etc.) as in EnergyPlus, and compares the tank states (e.g., SOC) and the calculated outlet conditions (e.g., outlet temperature) with those in EnergyPlus. Figure 7 shows the comparison results for SOC and outlet temperature in discharging mode. Figure 8 shows the comparison results in charging mode. The comparisons indicate that the simulated SOC and outlet temperature of the IST Modelica model are in excellent agreement with the outputs of the IST EnergyPlus model.

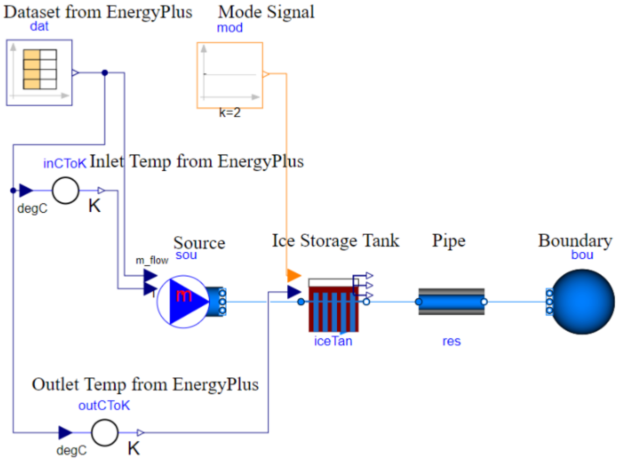


Figure 6. Virtual testcase for the IST Modelica model.

Three statistical metrics (CV(RMSE), NMBE, and R^2) were calculated to evaluate the accuracy of the prediction, and the results are shown in Table 5. R^2 ranges from 0.9311 to 0.9999, CV(RMSE) ranges from 0.00 % to 0.55 %, and NMBE is 0.00 % in all scenarios. All three metrics show excellent agreement between the tank performance predictions from the IST Modelica model and the IST EnergyPlus model, which is not a surprise since these two models use the same mathematical equations, though our IST model has more detailed local controls.

4.2 Simulated Data vs. Measured Data

In this section, we validate the model prediction with the experimental data obtained from an ice tank testbed at NIST (Pertzborn 2016, Pradhan 2020). Per the manufacturer, the ice storage tank at NIST contains 3,105 L of water and when fully frozen the ice has a capacity of 274 kWh, designed to be discharged over an eight-hour period with an inlet temperature of 10 °C. The chilled water that flows through the IST is a 30 % PG and 70 % water solution, and the heat exchanger inside the IST is a spiral wound polyethylene tube. The data was collected at a 0.10 Hz rate. The measured temperature and flow rate of the chilled water are used as the boundary conditions in the Modelica virtual testcase. Table 4 shows the polynomial coefficients for discharging mode and charging mode, which are obtained by regression of the measured data.

Table 4. Polynomial coefficients of curve fitting data.

Coefficients	d_1/c_1	d_2/c_2	d_3/c_3	d_4/c_4	d_5/c_5	d_6/c_6
Discharging	5.54E-5	-1.46E-4	9.28E-5	1.12E-3	-1.10E-3	3.01E-4
Charging	2.00E-4	0	0	0	0	0

Table 5. Results of statistical metrics (Benchmark: EnergyPlus).

Mode	*SOC*			*Outlet Temperature*		
	CV(RMSE)	R^2	*NMBE*	*CV(RMSE)*	R^2	*NMBE*
Discharging	0.35 %	0.9999	0.00 %	0.00 %	0.9999	0.00 %
Charging	0.55 %	0.9986	0.00 %	0.04 %	0.9311	0.00 %

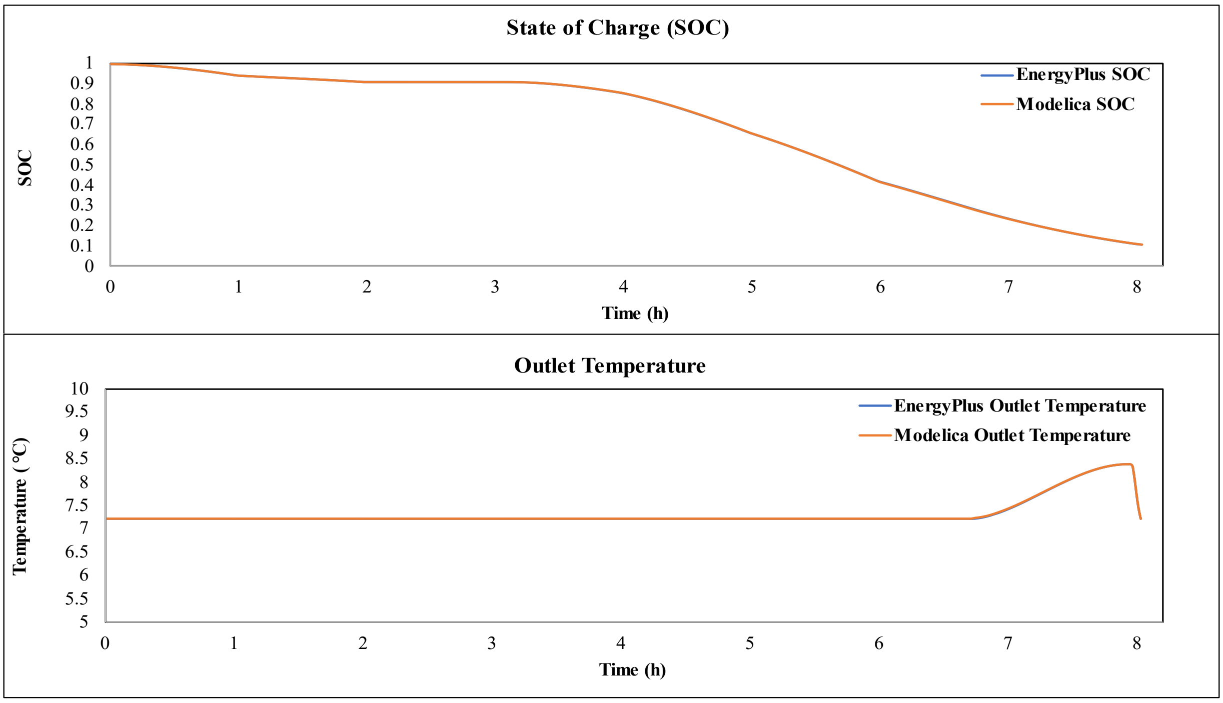


Figure 7. Discharging results comparison of the Modelica model with the EnergyPlus model.

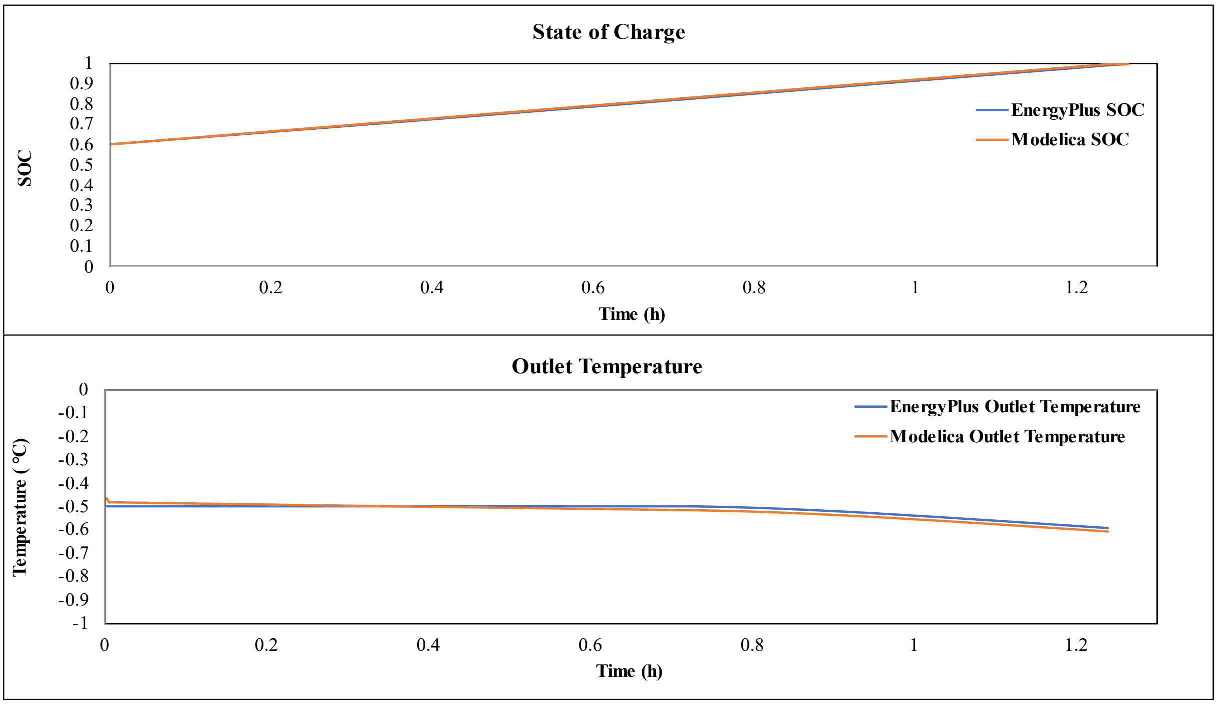


Figure 8. Charging results comparison of the Modelica model with the EnergyPlus model.

DOI
10.3384/ecp21181177

Table 6. Results of statistical metrics (Benchmark: measured data from NIST).

Mode & Date	*SOC*			*Outlet Temperature*		
	CV(RMSE)	*R^2*	*NMBE*	*CV(RMSE)*	*R^2*	*NMBE*
Discharging (5/14/2018)	7.09 %	0.9778	0.27 %	0.27 %	0.8281	0.21 %
Charging (5/16/2018)	10.20 %	0.9810	0.44 %	0.10 %	0.8344	0.03 %

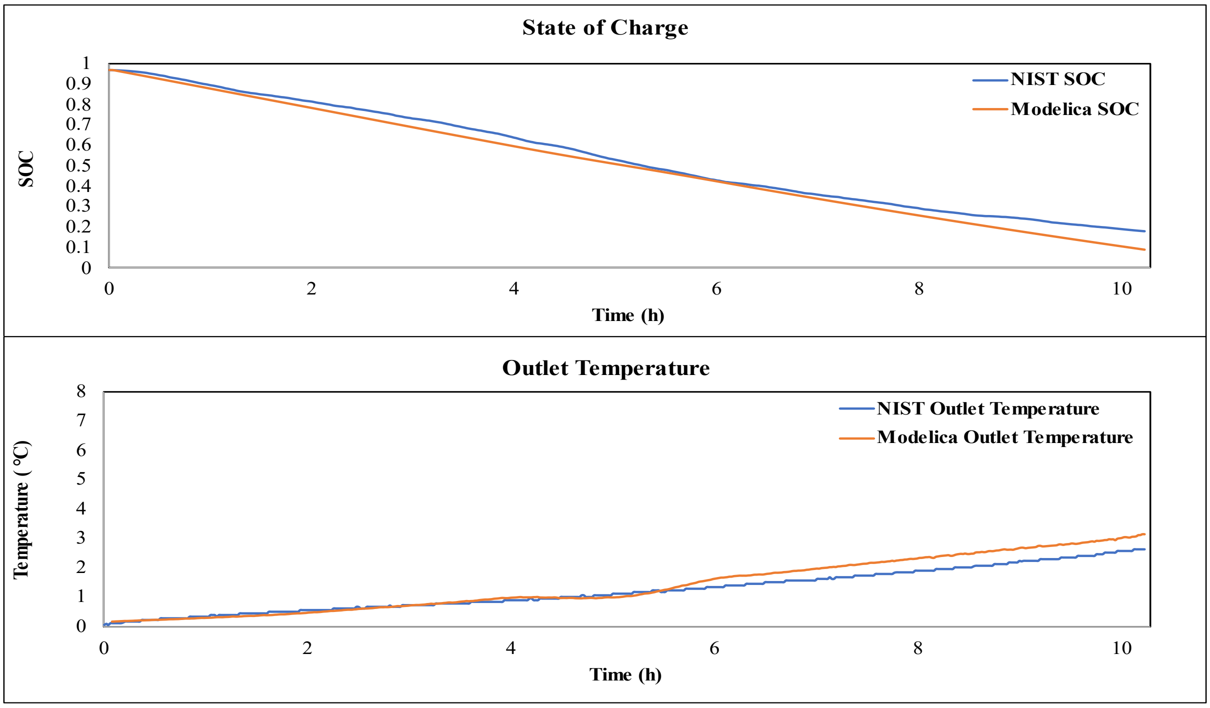


Figure 9. Discharging results comparison of the simulated data with the experimental data on 5/14/2018.

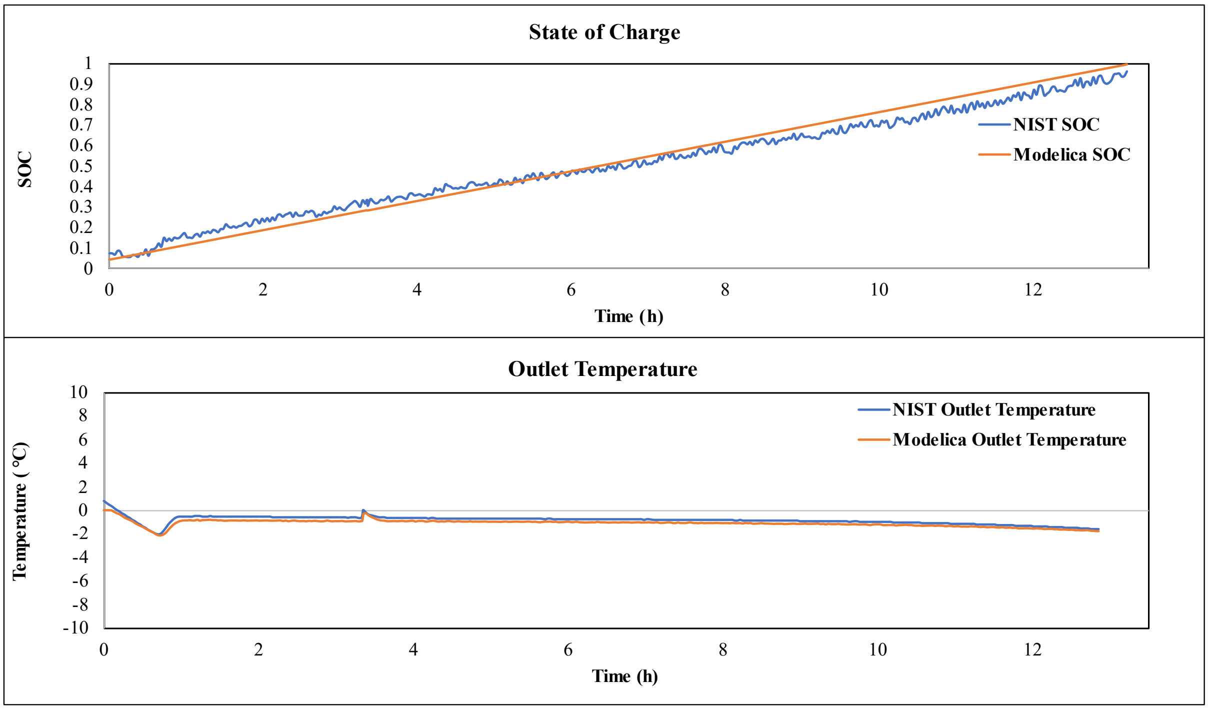


Figure 10. Charging results comparison of the simulated data with the experimental data on 5/16/2018.

For the discharging mode, the Modelica model is validated using the experimental data on 5/14/2018. For the charging mode, the Modelica model is validated using the experimental data on 5/16/2018. Figure 9 and Figure 10 show the comparison of simulated results with measured data of the SOC and outlet temperature for two days, respectively.

Table 6 shows the results of three accuracy metrics (CV(RMSE), NMBE, and R^2). The R^2 (0.8281 - 0.9810) values are high enough to indicate good agreement between the predictions and the measurement data. ASHRAE Guideline 14 suggests that the predicted model shall have a CV(RMSE) up to 30 % and an NMBE up to 10 % using hourly calibration data (ASHRAE 2014). Comparing the IST Modelica model with the experimental data from NIST, the results of CV(RMSE) (0.10 % - 10.20 %) and NMBE (0.03 % - 0.44 %) indicate that the IST Modelica model can provide good accuracy according to ASHRAE Guideline 14.

5 Conclusion

This study implemented an ice storage tank model based on the Modelica Buildings Library. The model was then compared to and validated against the EnergyPlus model and the measured data from a real ice tank system, respectively. The validation results quantified by three statistical metrics show a good prediction accuracy. In the future, the proposed IST Modelica model will be tested on system-level control evaluations and be used for load side management for better building-to-grid integration.

Acknowledgements

The research reported in this paper was partially supported by the Building Technologies Office at the U.S. Department of Energy through the Emerging Technologies program under award number DE-EE0009153.

Nomenclature

Abbreviations:

ASHRAE: American Society of Heating, Refrigerating and Air-Conditioning Engineers
CV(RMSE): Coefficient of Variation of Root Mean Square Error
IST: Ice Storage Tank
LMTD: Logarithmic Mean Temperature Difference
MBL: Modelica Buildings Library
NIST: National Institute of Standards and Technology
NMBE: Normalized Mean Bias Error
PG: Propylene Glycol
PI : Proportional-Integral
SOC: State of Charge

Symbols:

c_i: Charging coefficients
d_i: Discharging coefficients
H_f: Latent heat of fusion for water at 0 °C
$\dot{m}$: Mass flow rate of liquid
m_{ice}: Ice mass in the tank
P_{ch}: Charged fraction
$\dot{q}$: Instantaneous heat transfer rate
$\dot{q}^*$: Normalized instantaneous heat transfer rate
Q_{sto}: Total latent storage capacity
T_{in}: Tank inlet temperature
T_{out} : Tank outlet temperature
T_{fre}: Freezing temperature of the water
Δt: Timestep of the operation data used in the curve fitting
ΔT_{lm}: LMTD
ΔT_{lm}^*: Normalized LMTD
ΔT_{nom}: Nominal temperature difference

References

Beckman, William A., Lars Broman, Alex Fiksel, Sanford A. Klein, Eva Lindberg, Mattias Schuler, and Jeff Thornton (1994). "TRNSYS The most complete solar energy system modeling and simulation software". In: *Renewable energy*, 5(1-4), pp.486-488. DOI: 10.1016/0960-1481(94)90420-0.

Braun, James E. (1990). "Reducing energy costs and peak electrical demand through optimal control of building thermal storage". In: *ASHRAE transactions*, 96(2), 876-888. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.164.579&rep=rep1&type=pdf.

Candanedo, J. A., V. R. Dehkordi, and M. Stylianou (2013). "Model-based predictive control of an ice storage device in a building cooling system". In: *Applied Energy*, 111, pp.1032-1045. DOI: 10.1016/j.apenergy.2013.05.081.

Crawley, Drury B., Linda K. Lawrie, Frederick C. Winkelmann, Walter F. Buhl, Y. Joe Huang, Curtis O. Pedersen, Richard K. Strand et al. (2001). "EnergyPlus: creating a new-generation building energy simulation program". In: *Energy and buildings*, 33(4), pp.319-331. DOI: 10.1016/S0378-7788(00)00114-6.

Fu, Yangyang, Michael Wetter, and Wangda Zuo (2018). *Modelica models for data center cooling systems*. University of Colorado Boulder. URL: https://www.osti.gov/servlets/purl/1479111.

Fu, Yangyang, Wangda Zuo, Michael Wetter, James W. VanGilder, and Peilin Yang (2019). "Equation-based object-oriented modeling and simulation of data center cooling systems". In: *Energy and Buildings*, 198, 503-519. DOI: 10.1016/j.enbuild.2019.06.037.

Fu, Yangyang, Xing Lu, and Wangda Zuo (2019). "Modelica models for the control evaluations of chilled water system with waterside economizer". In: *Proceedings of the 13th International Modelica Conference*, Regensburg, Germany, March 4–6, 2019.URL:https://2019.international.conference.modelica.org/proceedings/html/papers/Modelica2019paperP09.pdf.

Guideline, A.S.H.R.A.E. (2014). *Measurement of energy, demand, and water savings*. ASHRAE Guidel 14-2014. URL: https://upgreengrade.ir/admin_panel/assets/images/books/ASHRAE%20Guideline%2014-2014.pdf.

Henze, Gregor P. and Jobst Schoenmann (2003). "Evaluation of reinforcement learning control for thermal energy storage systems". In: *HVAC&R Research*, 9(3), 259-275. DOI: 10.1080/10789669.2003.10391069.

Ihm, Pyeongchan, Moncef Krarti, and Gregor P. Henze (2004). "Development of a thermal energy storage model for EnergyPlus". In: *Energy and Buildings*, 36(8), 807-814. DOI: 10.1016/j.enbuild.2004.01.021.

Jekel, Todd Bryant (1991). "Modeling of ice-storage systems". Master's thesis. University of Wisconsin–Madison, URL: https://minds.wisconsin.edu/bitstream/handle/1793/45602/Jekel1991.pdf?sequence=1.

Mattsson, Sven Erik, Hilding Elmqvist, and Martin Otter (1998). "Physical system modeling with Modelica". In: *Control Engineering Practice*, 6(4), 501-510. DOI: 10.1016/S0967-0661(98)00047-1.

Pertzborn, Amanda J. (2016). *Intelligent Building Agents Laboratory: Hydronic System Design*. US Department of Commerce, National Institute of Standards and Technology. URL:https://nvlpubs.nist.gov/nistpubs/TechnicalNotes/NIST.TN.1933.pdf.

Pradhan, Ojas, Amanda Pertzborn, Liang Zhang, and Jin Wen (2020). "Development and Validation of a Simulation Testbed for the Intelligent Building Agents Laboratory (IBAL) using TRNSYS". In: *ASHRAE Transactions, 126*, pp.458-466.

Strand, Richard Karl (1992). "Indirect ice storage system simulation". M.S. Thesis, Department of Mechanical and Industrial Engineering, University of Illinois at Urbana-Champaign.

Wetter, Michael (2011). "A View on Future Building System Modeling and Simulation". United States. URL: https://www.osti.gov/servlets/purl/1050665.

Wetter, Michael, Wangda Zuo, Thierry S. Nouidui, and Xiufeng Pang (2014). "Modelica buildings library". In: *Journal of Building Performance Simulation*, 7(4), pp.253-270. DOI: 10.1080/19401493.2013.765506.

Status of the TransiEnt Library: Transient Simulation of Complex Integrated Energy Systems

Anne Senkel[1] Carsten Bode[1] Jan-Peter Heckel[2] Oliver Schülting[3]
Gerhard Schmitz[1] Christian Becker[2] Alfons Kather[3]

Hamburg University of Technology, Am Schwarzenberg-Campus 1, Hamburg, Germany
[1]Institute of Engineering Thermodynamics, {anne.senkel, c.bode, schmitz}@tuhh.de
[2]Institute of Electrical Power and Energy Technology, {jan.heckel, c.becker}@tuhh.de
[3]Institute of Energy Systems, {oliver.schuelting, kather}@tuhh.de

Abstract

The open-source Modelica library TransiEnt Library was developed within the research project TransiEnt.EE. This paper presents two major library extensions which were developed in the follow-up research project ResiliEntEE. Modeling of the power sector on transmission grid level is now possible due to the implementation of the complex bus voltage. In the gas sector, the efficiency of mass and energy balance computation was improved. Furthermore, an efficient physical pressure loss model was added leading to more realistic results and faster simulations. One possible application of the library is presented in an exemplary simulation of an integrated energy system. It is shown that the dynamic simulation allows the representation of disturbances and their possible consequences in coupled sectors. Thus, next to cost and CO_2 emissions, an integrated energy system can be assessed in terms of its resilience as well.

Keywords: Integrated Energy Systems, Dynamic Interactions, Electricity, Gas, Heat, TransiEnt Library

1 Introduction

The energy sector is facing severe changes, not only in the upcoming years but already today. In its recent study, Agora Energiewende identifies ten megatrends that will influence the development of the future energy system (Agora Energiewende 2019). Among them are:

- Decarbonization,
- Cost regression for renewables,
- Digitalization,
- Electrification,
- Urbanization,
- Decentralization, and
- Interdependence.

Each of these tendencies will not only lead to a more sustainable but also more complex and dynamic integrated energy system (IES). In Figure 1, typical components of an IES are depicted for the sectors gas, heat, and power. Researchers and decision-makers alike are facing the challenge of identifying different development paths and their advantages and disadvantages. To be able to analyze and evaluate these, corresponding models need to be developed and the complex and dynamic behavior of the considered IES must be implemented. In terms of evaluation, it is notable that not only cost and CO_2 emissions are of interest but resilience is gaining importance as well (Linkov and Palma-Oliveira 2017).

In this context, Modelica offers support for multi-domain dynamic simulation and, thus, the simulation of different energy sectors within a single, coherent model. Therefore, multi-disciplinary approaches are supported, as are necessary to develop sophisticated models for each sector. Moreover, the object-orientation allows the development of libraries with user-friendly, scalable models that offer a high degree of reusability and adaption.

Based on these considerations, the TransiEnt Library has been developed in the TransiEnt.EE and ResiliEntEE projects (Hamburg University of Technology 2020). Using it, users may implement dynamic interactions in their energy system models and use the results for further assessments.

After a brief presentation of the projects TransiEnt.EE and ResiliEntEE as well as their major outcome, the TransiEnt Library, this paper presents the main extensions of the library in the gas and power sector which were realized in the ResiliEntEE project. To provide insights into the possibilities of the TransiEnt Library, an exemplary model of an IES is presented and its simulation results are discussed.

1.1 TransiEnt.EE

The TransiEnt.EE research project was conducted at the Hamburg University of Technology from 2013 to 2017 (Andresen et al. 2017). Its main goal was to identify innovative and reliable ways to efficiently integrate renewable energies into existing urban energy systems to maximize the energy system's self-sufficiency. For their analysis, the researchers focused on the city of Hamburg's energy supply system. In this context, the TransiEnt Library was developed to not only allow the modeling necessary for the concrete research project but also for future studies and related research interests (Andresen et al. 2015).

In TransiEnt.EE, the considered energy systems were analyzed according to their cost and CO_2 emissions, enabling efficiency assessments of the considered scenarios with respect to the goals of the German energy transition.

DOI
10.3384/ecp21181187

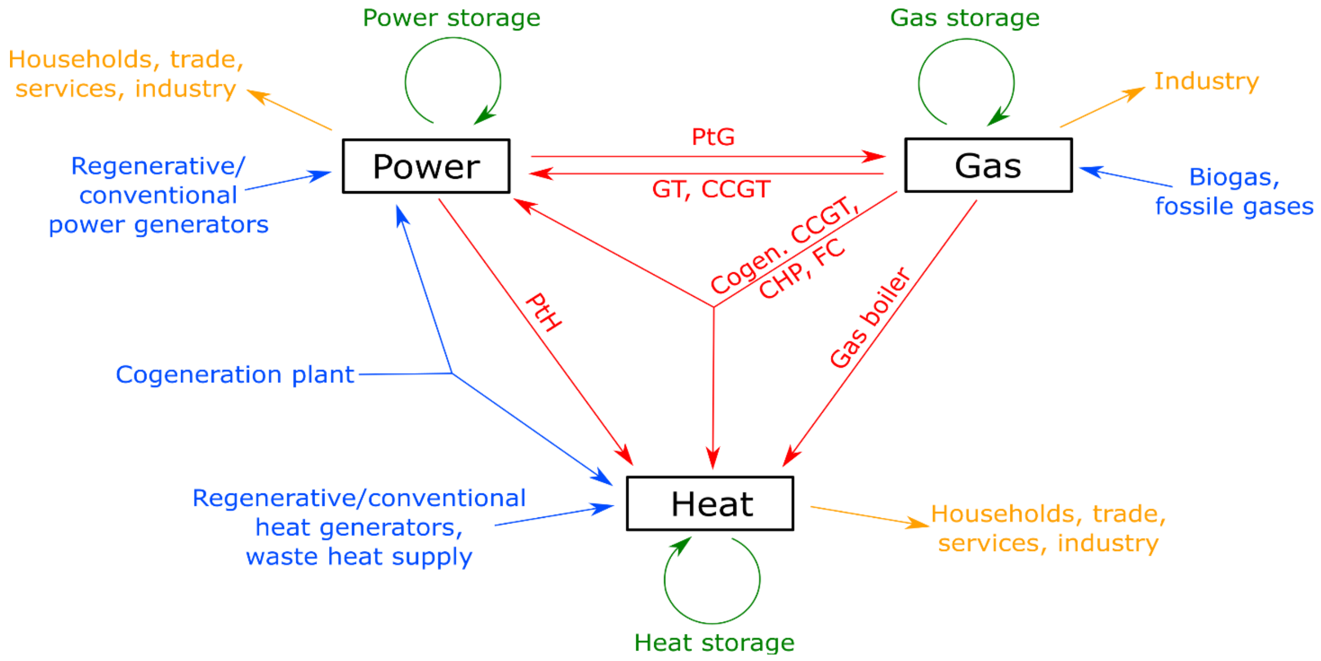

Power storage
Gas storage
Households, trade, services, industry
Industry
Power
Gas
PtG
GT, CCGT
Regenerative/ conventional power generators
Biogas, fossile gases
Cogen. CCGT, CHP, FC
PtH
Gas boiler
Cogeneration plant
Heat
Regenerative/conventional heat generators, waste heat supply
Households, trade, services, industry
Heat storage

Figure 1. Presentation of the three Sectors (black) with Energy Producers (blue), Conversion Technologies (red), Storages (green) and Consumers (yellow) (PtG: Power-to-Gas, GT: Gas Turbine, CCGT: Combined Cycle Gas Turbine, Cogen. CCGT: Cogeneration Combined Cycle Gas Turbine, CHP: Combined Heat and Power, FC: Fuel Cells, PtH: Power-to-Heat

1.2 ResiliEntEE

ResiliEntEE is the follow-up project to TransiEnt.EE conducted at the Hamburg University of Technology from 2017 until 2021. The project used the TransiEnt Library and aimed at refining it twofold:

First, the considered system was enlarged from the city of Hamburg to Northern Germany (including the federal states of Bremen, Hamburg, Lower Saxony, Mecklenburg-Western Pomerania, and Schleswig-Holstein). Towards this, a numerically efficient modeling approach needed to be found to enable fast model generation of the region considering different levels of aggregation and detail. Additionally, the gas and the electric grid modeling needed to be expanded to this superregional scale. Finally, a suitable control for the overall system was designed including the operational planning of renewable power producers, energy conversion technologies, and storage plants as well as the provision of system services and reactions to disruptions.

The second project aim was to use the TransiEnt Library models to evaluate the resilience of the energy system and its subsystems. Therefore, the existing models needed to be refined to show the behavior of the considered systems before, during, and after disturbances. For this purpose, a special focus was laid on the electric grid, expanding the models to spatially resolved, stationary and dynamic computations (Heckel and Becker 2019). Furthermore, the connections between the sectors were revised to be able to answer the questions of how disruptions in one sector affect other sectors.

Finally, a quantitative evaluation method was developed to assess the resilience of energy systems using dynamic simulation results (Senkel, Bode, and Schmitz 2021, 2019). This enables the determination of the resilience of future energy supply systems against stressors such as component failure, weather extrema, control errors, and others. This enables the comparison of different possibilities towards improving system resilience.

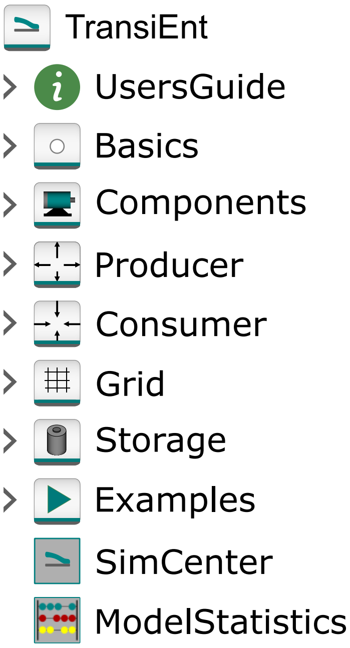
1.3 The TransiEnt Library

The TransiEnt Library provides models for typical energy system components, including energy producers, consumers, storage units, conversion plants, and grid components (Hamburg University of Technology 2020). The resulting library structure is depicted in Figure 2. A detailed introduction to a preliminary version of the TransiEnt Library is given by Andresen et al. (2015).

Using the TransiEnt Library, the fluctuating energy provision of renewable energies can be modeled dynamically, allowing investigations into their influence on the energy system. Due to the dynamic simulation, dynamic interactions between the sectors and their impact on the flexibility and stability of the overall system can be depicted. This allows sensitivity analyses and the evaluation of different integration and control strategies. Owed to the multi-domain approach of Modelica, this can be done in one model covering all sectors, allowing a detailed analysis of the overall system. One example is the investigation of excess power utilization in the heat sector (Bode and Schmitz 2019).

Towards deduplication, the TransiEnt Library uses components of the ClaRa Library (Hamburg University of Technology, TLK-Thermo GmbH, and XRG Simulation GmbH 2021), the TILMedia Library (TLK-Thermo GmbH and Institut für Thermodynamik, Technische Universität Braunschweig 2021), and the Buildings Library (Wetter et al. 2014).

The ClaRa Library allows the modeling of the transient thermal behavior of power plants and related systems. The TransiEnt Library reuses basic components from that library, including but not limited to pipes, heat exchangers, and valves. Furthermore, certain components of the gas sector are based on



TransiEnt
UsersGuide
Basics
Components
Producer
Consumer
Grid
Storage
Examples
SimCenter
ModelStatistics

Figure 2. Structure of the TransiEnt Library

models from the ClaRa Library (see section 2.2), for example, pipe and valve models.

The TILMedia Library provides thermophysical properties for various fluids and solids. The TILMedia Library models provide better numerical performance and robustness compared to Modelica.Media (Modelica Association 2020). However, coupling of models using the different media models is possible and a suitable adapter model is included in the TransiEnt Library.

From the Buildings Library, models for the heat transfer within walls and their sky radiation exchange as well as the air volume within a building are used to provide a low-order heat consumer model and thus the modeling of cool-down and heat-up processes in buildings connected to the considered energy system.

2 New Developments and Components

As a result of the ResiliEntEE project, several new developments and models were implemented in the TransiEnt Library. Especially the gas and power sector modeling were enhanced, as described in the following section. Further developments and results are documented by Bode et al. (2021).

2.1 Modeling of the Power Sector

Electric power systems can be modeled on different levels. On the one hand, the focus of dynamic modeling can vary, and, on the other hand, it is possible to differentiate between the levels of modeling the grid connections. In terms of simulation, the simplest option is a steady-state simulation referred to as load flow calculation. In load flow calculations, the grid state is calculated in the form of complex bus voltages based on the representation of the alternating voltages and currents as quasi stationary root mean square (RMS) phasors. The grid state is influenced by the load and generation as well as the switch settings in the electric grid.

To focus on the dynamics, the RMS simulation, also based on phasors, can be used. The frequency of the electric voltage is chosen as an additional variable. This approach allows the use of numerical integrators with variable step sizes above 20 ms in 50 Hz systems, enabling efficient simulations of entire years. The RMS simulation approach can assess the electromechanical processes and the behavior of the controllers that ensure the operation of the electric grid.

Lastly, the electric energy system can be simulated using the electromagnetic transient simulation approach. In this approach, voltages and currents are time-dependent including their sinusoidal oscillation. The electromagnetic transient simulations require numerical integrators with step sizes below 20 ms, which have high numerical cost.

In the TransiEnt Library, the RMS approach is chosen. This allows numerically efficient simulations since the stiffness of the problem of the IES simulation can be reduced. Stiff problems occur when time constants from a large range of orders of magnitude are part of the problem and the numerical integrator needs to adapt the step size to the lowest time constant. This is the case for IES simulations when the electric subsystem is simulated using the electromagnetic transient approach. Processes in the heat and gas sector have higher time constants starting at approximately a few seconds, as shown in Figure 3, which gives an overview of time constants of different processes from the power, heat, and gas sector. In addition, the time intervals of the stability phenomena in electric power systems are given. The relevant stability phenomena of the electric subsystem in IES are frequency and voltage stability.

With the RMS simulation approach of the electric energy system, time constants starting at approximately 100 ms are regarded. The RMS simulation of electric power systems becomes steady state when neither load and generation are changed nor disturbances occur. Using appropriate models for the IES simulation within the RMS approach, problems' stiffness can be reduced. Hence, processes in the IES with time constants, starting at approximately one second, are considered. In contrast, electromagnetic transient simulations always consider time constants of 20 ms and lower.

The different levels of modeling the grid connections can be demonstrated through the evolution of electrical interfaces in the TransiEnt Library. In the TransiEnt.EE project, the electric grid was modeled as a copper plate with ideal interconnections. This modeling is based on the `ActivePowerPort` interface which contains the active power as a flow variable and the frequency of the electric grid as a potential variable. With this interface, all electrical components modeled in a simulation model are connected with each other at one bus bar.

In the next step, the modeling was extended to the modeling of radial grid structures as these structures often occur in low voltage grids. For this approach, the `ApparentPowerPort` was introduced. This interface also considers the electric bus voltage magnitude as a potential variable and the reactive power as a flow variable.

In the ResiliEntEE project, interconnected electric power systems, as they can be found on the transmission grid level, are modeled. Due to this extended scope, a new electrical interface was introduced (Heckel and Becker 2019): The `ComplexPowerPort` adds the complex bus voltage angle as a potential variable to the modeling. Thus, the complete complex bus voltage is part of the interface leading to the appropriately named `ComplexPowerPort`.

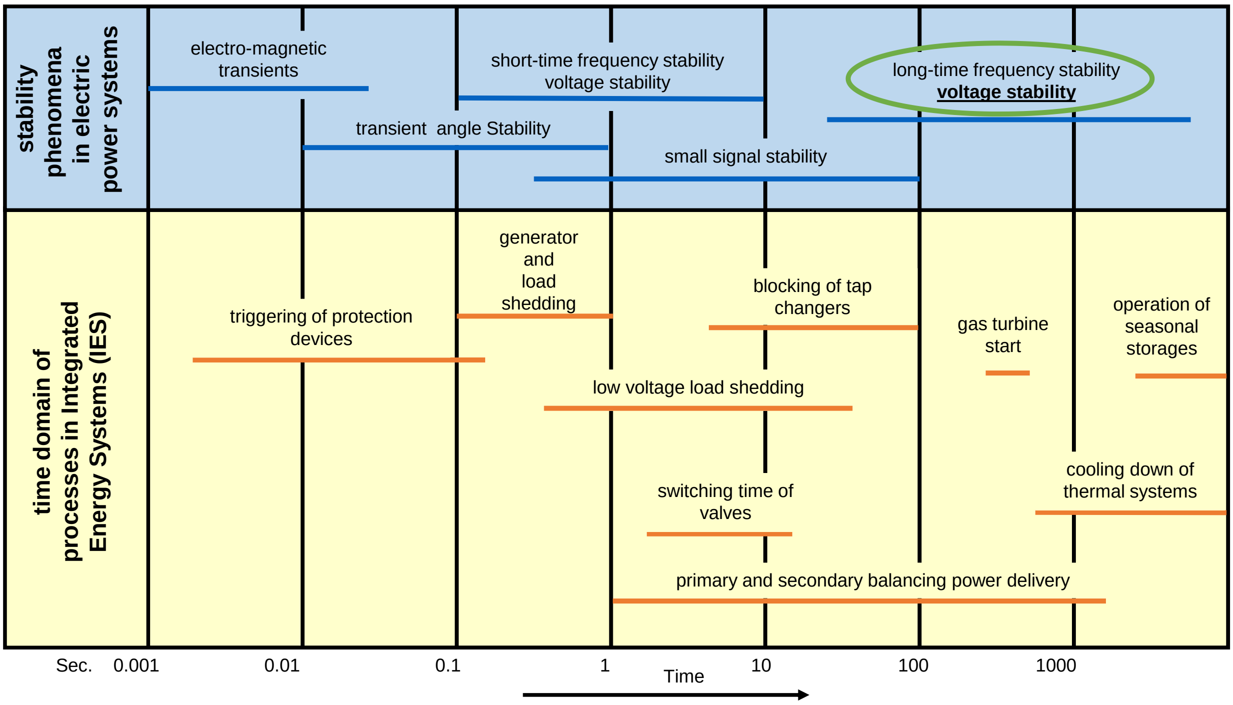

stability phenomena in electric power systems
electro-magnetic transients
short-time frequency stability voltage stability
long-time frequency stability voltage stability
transient angle Stability
small signal stability
time domain of processes in Integrated Energy Systems (IES)
generator and load shedding
triggering of protection devices
blocking of tap changers
gas turbine start
operation of seasonal storages
low voltage load shedding
switching time of valves
cooling down of thermal systems
primary and secondary balancing power delivery
Sec.
0.001
0.01
0.1
1
Time
10
100
1000

Figure 3. Overview of time constants of processes in IES and the corresponding stability phenomena in electric grids

At two flow variables and three potential variables, the interface is over-determined. The frequency is the additional variable that is part of the modeling due to the RMS simulation approach. For the problem of overdetermined interfaces, the Modelica Specification Section 9.4 gives a workaround (Modelica Association 2017). This workaround is followed here: A custom frequency type with an `equalityConstraint` is defined for the electric frequency of the grid. With the definition of grid busses as "potential roots" and the grid connection components as "branches" of a "connection tree", this approach allows the RMS modeling of electric power systems with the `ComplexPowerPort`. All mentioned electrical interfaces in the TransiEnt Library are unipolar allowing flexible modeling in a clearly arranged way.

With the introduction of the `ComplexPowerPort`, new component models have been added to the TransiEnt Library. These models can be divided into steady-state and dynamic components. As steady-state components, the different bus types, which are considered in the modeling of electric energy systems, are implemented as boundary conditions. Boundary conditions set variables at the interface to given values allowing other variables to take arbitrary values. The bus types are:

- Slack Bus: Voltage magnitude and angle are given; the frequency is given for steady-state calculation and the other quantities are calculated.
- PV Bus: Active power and voltage magnitude are fixed; other quantities are calculated. This bus model is used for power plants with voltage control.
- PQ Bus: Active and reactive power are given; other quantities are calculated. PQ Busses mainly represent loads.

Transmission line models based on the pi network representation in two-port representation model the grid connections. The transmission line model was validated with different commercial and open-source steady-state tools. For the dynamic simulation, these models can be reused because the grid connections are assumed steady-state in the RMS simulation of electric energy systems. Additionally, different transformer models allow the modeling of interconnected and spacious electric transmission systems.

As dynamic component models, new models for synchronous generators have been implemented. The TransiEnt Library models allow the modeling of synchronous machines at different levels of detail. All models regard the electromechanical behavior but differ in the electric sub model, which links the voltage magnitude and the active as well as reactive power.

The models can be split into one- and two-axis models. One-axis models are suitable for the description of cylindrical rotor machines. Moreover, the two-axis description, based on the Park Transformation (Park 1929), allows a more general description of non-symmetric electric machines.

The different levels of detail enable the adaption of the modeling to the simulation scope. Thus, the trade-off between high accuracy and affordable simulation costs can be resolved in the context of each problem to be simulated.

In addition to the synchronous machines, different motor models, such as an induction motor, have been added to the TransiEnt Library. These models allow a more detailed consideration of the integration of the power sector with the other sectors of an IES through conversion plants.

2.2 Modeling in the Gas Sector

The TransiEnt Library contains all models necessary to simulate future gas grids. An example model is shown in
Figure 4 that depicts how the components can be connected. Required fluid properties are supplied by the TILMedia Library.

The essential components of a gas grid are the pipes transporting the gas. The respective model is based on the ClaRa Library and uses the finite volume method for one-dimensional flow. Each control volume contains an overall mass balance, component mass balances, and an energy balance. The heat transfer can be modeled using different heat transfer models inside the pipe and additional wall and external heat transfer models.

To work more efficiently in large networks, different simplifications were added for the model in the TransiEnt Library: It is possible to use isothermal flow, which is a common assumption in gas grids (Cerbe and Lendt 2017) to eliminate the specific enthalpy as a state.

In addition, the model was augmented with simplified component mass balances: If the gas in the grid has a constant composition, the mass fractions are eliminated from the system of equations. Furthermore, if, for example, natural gas with constant composition is transported in the grid and hydrogen is admixed, the hydrogen fraction is the only truly variable component and all others depend on it. In this case, the dependent fractions can be calculated by scaling the nominal composition of the main gas accordingly. This also reduces the number of states.

In the ClaRa Library, all pressure loss models are either linear or quadratic and depend on nominal values. To enable simpler pipe parametrization and gain more realistic results, a pressure loss calculation based on Cerbe and Lendt (2017) is implemented by adapting the equations from Modelica.Fluid.Dissipation (Modelica Association 2020): Pipe discretization can be avoided by using the pressure loss equation for a compressible fluid under the assumption that the quadratic mean pressure equals the arithmetic mean pressure. This leads to a small error but also efficient simulation since the equation does not contain quadratic pressures and each pipe can consist of only one control volume. The required dynamic viscosity is not supplied by the TILMedia Library, so linear correlations based on media data from REFPROP (Huber et al. 2018) for natural gas, methane-hydrogen mixtures, and pure hydrogen are used.

In addition to pipes, junctions are an important part of the gas grid since they enable the mixing of different gas flows. Additionally, they connect pipes and their corresponding pressure losses and volumes in a numerically efficient way. The junction model is also based on a model from the ClaRa Library and consists of a volume with an arbitrary number of gas ports. The volume contains the same equations as the control volume of a pipe with the same simplifications for component mass as well as energy balances. Linear pressure losses between the volume and each gas port can be added to avoid nonlinear systems of equations in certain cases.

To balance gas supply and demand, gas storage units are necessary. The TransiEnt Library contains different storage models which differ in their level of detail. First, the gas has either constant or variable composition. Second, the gas storage either contains a fluid model, with which temperature and pressure in the gas storage can be calculated, or simply uses a constant pressure. The latter is the faster model but neglects the gas state in the storage. The model containing a fluid model is similar to the junction model but different heat transfer models, typical for certain kinds of gas storage, can be used: cavern storage, gas pressure vessel, or adiabatic storage. Furthermore, the pipes, through which the gas enters and leaves the storage, can be included in the model: either using a pressure loss and heat transfer model or neglecting pressure losses and assuming constant outlet temperatures.

Furthermore, suitable boundary conditions are needed to simulate a gas grid. The TransiEnt Library contains several basic boundary conditions that always set three values:

- pressure, mass flow rate, volume flow rate at standard conditions, or enthalpy flow rate
- mass or molar fraction, and
- temperature or specific enthalpy.

Gas consumers also act as boundary conditions for the gas grid. Different kinds of gas-fired heat producers, for example, gas boilers, gas heat pumps, and combined heat and power plants, as well as gas-fired power plants, for example, gas turbines and combined-cycle power plants, are part of the TransiEnt Library.

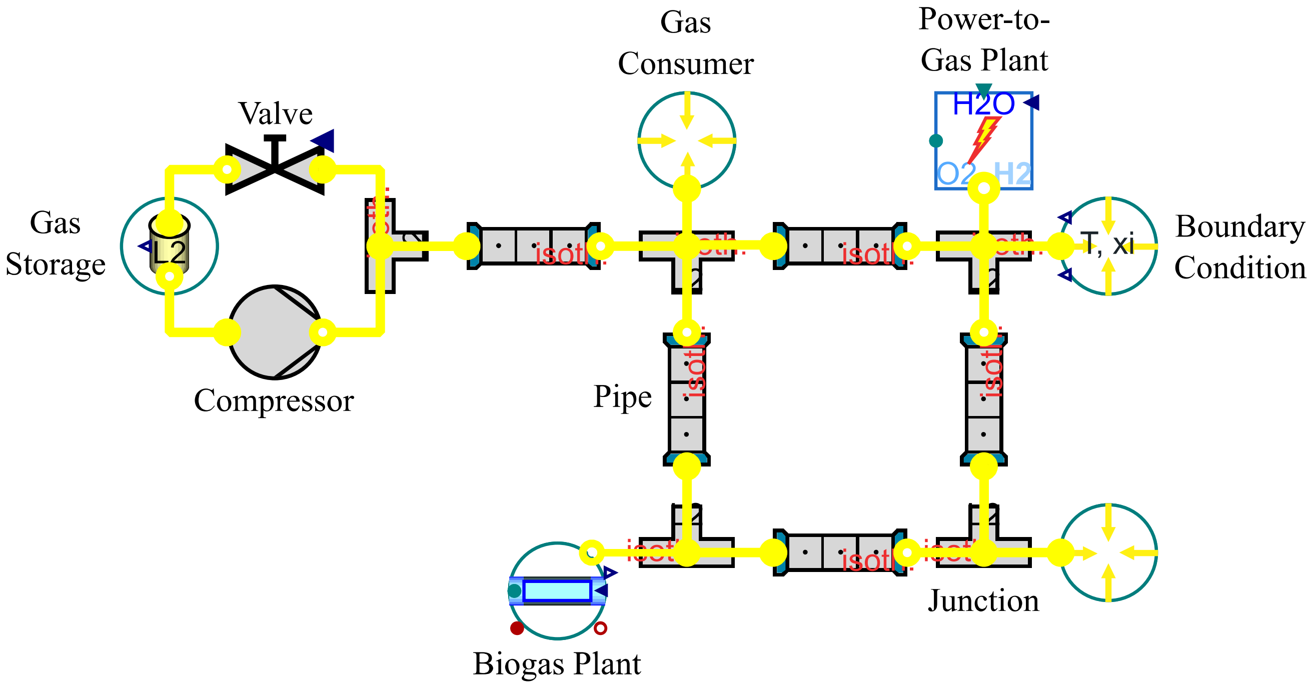


Figure 4. Gas Model with New Components

Major gas-producing units of the future will be biogas and Power-to-Gas plants. In a biogas plant, different organic substrates are transformed into biogas using bacteria. The model in the TransiEnt Library contains the ADM1 model (Klimiuk et al. 2015), which models the transformation of different organic components. Moreover, correlations for the power demand of different stirrers as well as the heat transfer from the heater to the reactor and from the reactor to the ambient are included.

The second technology, Power-to-Gas, uses electricity to generate hydrogen via electrolysis. The core component, the electrolyzer, is either modeled using an efficiency curve or more detailed physics of the electrolyzer (Webster and Bode 2019). Both models include either quasi-stationary, first-order, or second-order dynamics as well as the possibility to give a certain amount of waste heat to a connected component.

In some Power-to-Gas plants, the hydrogen is transformed to synthetic natural gas using CO_2 to be fed directly into the natural gas grid. Different models are included in the TransiEnt Library which either use reaction kinetics, reaction equilibrium, or a constant conversion rate. The required CO_2 can be separated from air, modeled by a simple direct air capture plant model.

All parts of the Power-to-Gas plant can be combined to so-called feed-in stations which contain suitable controllers as well as storage units.

Also, the TransiEnt Library contains a model for a steam methane reformer including related reactor models, for example, a pre-reformer and a water-gas shift reactor. Additional minor components, which are part of the TransiEnt Library, are simple compressor, heat exchanger, and valve models.

3 Example of Use

The following chapter presents the `CoupledLargeScale` model from the Example Package of the TransiEnt Library. In this model, the sectors gas, heat, and power are integrated and coupled by a combined-cycle power (CCP) plant with heat extraction and an electrolyzer. The structure of the IES is depicted in Figure 5, the parameters of the system are listed in Table 1.

3.1 Model Description

In the gas sector, a gas grid containing one loop supplies gas consumers at three points in the grid (NW, NE, SE). In the northwest, a constant gas demand is modeled as it is often the case for industrial consumers. In the east, the gas is used to supply gas boilers for the heat supply of housing areas. Additionally, the CCP plant in the southeast burns gas to provide heat to a district heating network (DHN) and power to the electric grid. The gas is provided to the gas grid by a gas source in the southwest at a pressure of 12.5 bar. Moreover, the electrolyzer in the southwest uses excess power to produce hydrogen which is also fed into the gas grid.

The demand in the heating sector is modeled using a reduced-order approach considering the thermal capacities and conductances of the houses as well as thermal heat gains and ventilation losses. To achieve a reasonable computation time, the housing areas are modeled by computing the heat demand for one characteristic house and scaling it up accordingly. The heat is transported to the households in two different ways. The housing areas in the eastern part of the gas grid are supplied by gas boilers which are controlled in order to reach the set room temperature.

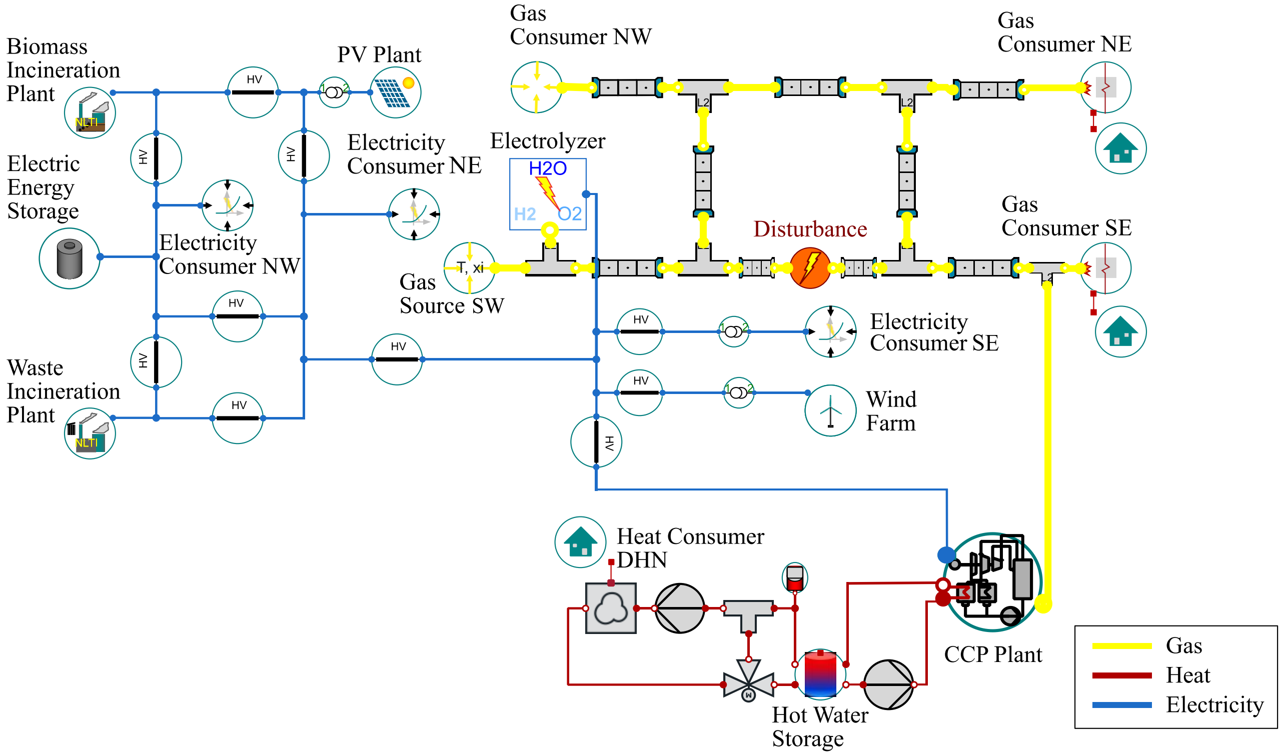


Figure 5. Graphical Layer of the Investigated Integrated Energy System

Table 1. Parametrization of the Investigated IES

Parameter	
Gas Sector	
Pressure Gas Source	12.5 bar
Minimum Pressure Gas Boiler	2 bar
Minimum Pressure CCP Plant	2 bar
Total Pipeline Length	47 km
Gas Grid Volume	4 400 m³
Heat Sector	
Number of Households NE	58 100
Number of Households SE	29 050
Number of Households DHN	6 260
Nominal Heat Flow Rate Gas Boiler NE	280 MW
Nominal Heat Flow Rate Gas Boiler SE	140 MW
Nominal Heat Flow Rate CCP Plant	30 MW
Buffer Storage Volume DHN	1 720 m³
Supply Temperature DHN	80 °C
Nominal Mass Flow Rate DHN	240 kg/s
Nominal Power Electric Boiler	140 MW
Efficiency Electric Boiler	0.95
Power Sector	
Nominal Power Biomass Incineration Plant	80 MW
Nominal Power Waste Incineration Plant	100 MW
Nominal Power Wind Park	60 MW
Installed PV Plant Area	500 m²
Nominal Power CCP Plant	60 MW
Nominal Power Electrolyzer	1 MW
Maximum Discharging Power Electric Energy Storage	50 MW
Maximum Charging Power Electric Energy Storage	70 MW
Storage Capacity Electric Energy Storage	6 GWh
Proportional Load Factor NW	2.6
Proportional Load Factor NE	2.5
Proportional Load Factor SE	0.8

The households in the south, however, are connected to their heat supply via the heating network. Hence, their heat supply is connected to the operation of the CCP plant which is not only controlled according to the current heat demand but also to the current electricity demand and production. Therefore, a hot water storage is placed between the heat consumer and the CCP plant, decoupling the two from each other. Thus, the heat supply of the consumers is ensured by a) controlling the supply temperature via a backmixing valve according to a heating curve depending on the ambient temperature and b) by controlling the mass flow rate through the heat exchanger at the consumer to reach the set room temperature. The heat output of the CCP plant is determined according to the set temperature of the top volume of the hot water storage and the physical constraints of the combined heat and power generation.

The electricity demand is modeled using SLP load profiles for agriculture, households, and industry (Price et al. 1993). Additionally, the frequency dependency of the consumers is modeled according to Price et al. (1993). The power generation units are biomass and waste incineration plants as well as a PV plant and a wind park, which are dependent on the solar irradiation and the wind velocity, respectively. To avoid curtailment of the PV plant and wind park, excess power is stored in an electric energy storage or used to produce hydrogen in the electrolyzer. The power provision is controlled according to the merit order: PV and wind plants – CCP plant – discharging the electric energy storage – garbage incineration plant – biomass incineration plant. For more detailed information including parametrization, refer to Senkel, Bode, and Schmitz (2021).

This energy system is modeled to be disturbed by the closure of the southern gas pipeline for 14 hours at the beginning of an exemplary February. At this time of the year, the lowest ambient temperatures and the lowest feed-in of renewable energy occur and therefore the strongest effects of the disturbances can be observed. The software used is Dymola (Dassault Systèmes 2021) with the Esdirk45a solver and the tolerance 10^{-6}. In the next section, the simulation results of the simulation with and without the induced disturbance are discussed.

3.2 Simulation Results

In Figure 6, the simulation results for the gas pressure, the aggregated enthalpy flow rate, the room temperature, the characteristic heat flow rate, the grid frequency and the electric power of the undisturbed and disturbed IES are depicted. The gas pressure is fluctuating since the gas demand within the gas grid varies as well, especially because of the ambient-temperature-dependent heat demand. It can also be noticed how the gas pressure decreases with increasing distance to the gas source as the pressure losses in the pipelines accumulate. When the gas pipeline in the south is closed, the pressure drops drastically, especially in the southeast since the distance through the gas grid increases the most. In this part of the grid, the controls of the CCP plant and the gas boiler react to this drop by shutting down the affected assets to protect them from too low pressures. Because of that, the gas demand in the grid decreases and the pressure recovers to a level around 8 bar. However, the original level of 10 to 12 bar is not reached until after the disturbance ceases. Here, the CCP plant and gas boilers are turned on with a one-hour time delay resulting in a small peak right after the pipeline is opened.

When looking at the enthalpy flow rate, the temperature dependence of the gas consumers in the east and, thus, their fluctuating gas demand becomes notable. In contrast to this, the gas consumer in the northeast has a constant gas demand. Since the nominal electric power of the CCP plant is twice as high as its maximum heat flow rate, its gas demand is dominated by the current power demand. Hence, its gas demand is constant except for low-load periods at night. Since only the CCP plant and the gas boilers are turned off, they are the only gas consumers for which the gas demand cannot be met during the disturbance.

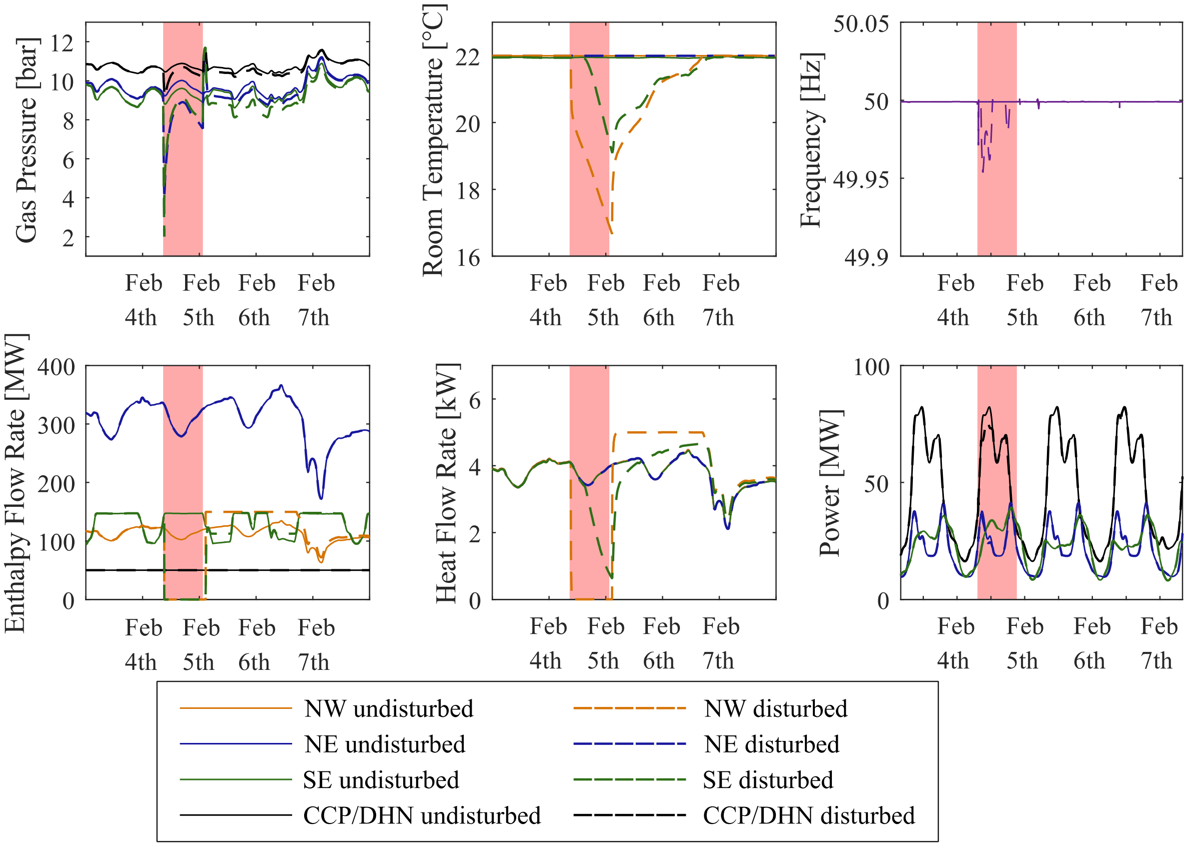


Figure 6. Simulation Results of the Disturbed and Undisturbed IES

The shutdown of the gas supply for the gas boilers and CCP plant also has consequences for the attached heat consumers. The shutdown of the gas boilers induces an immediate shutdown of the heat supply to the heat consumers in the southeast. Thus, their heat flow rate becomes zero and the houses cool down as seen in the drastic drop of the respective room temperatures.

The households supplied by the heating network do not cool down as fast as the southeastern ones. Here, the hot water storage is able to uphold the heat supply for several hours before it is completely discharged. After that, the heat supply also drops, alongside the room temperature.

After the disturbance, the houses need to be reheated. Therefore, the heat flow rate exceeds the heat demand in the undisturbed case. For the consumers in the southeast, the gas boiler is set to its maximum heat flow rate until the temperature recovers. In the DHN, the heat flow rate to the consumer is controlled differently. Moreover, the thermal inertia of the cooled water in the hot water storage leads to an additional delay in the provision of the maximum heat flow rate.

For the heat consumers in the northeast, the gas and, thus, the heat supply is not affected by the pipeline closure which is why their heat demand and set room temperature can be met at all times.

In the power sector, the other producers react to the shutdown of the CCP plant and the following frequency drop by providing balancing power. However, the demand is still higher than the supply. Hence, the frequency drops and reaches its minimum at 49.95 Hz. In response, the power consumers with the highest proportional load factors (NW and NE) react with an electric load reduction. When the demand becomes smaller, the power supply is sufficient enough to satisfy the demand of all power consumers. At this point, the frequency also returns to its set point of 50 Hz.

The presented simulation results show that a detailed analysis of a disturbed complex system is possible using dynamic simulation. The models of the TransiEnt Library offer detailed modeling and, thus, simulation of the necessary effects. Since this is also true for the energy conversion technologies, the consequences of the disturbance can not only be seen in the directly affected sector but also in the sectors coupled with it. Based on these simulation results, a quantitative assessment of the system's behavior is now possible. Senkel, Bode, and Schmitz (2021) describe a suitable approach and provide more information on the simulation of a system's resilience using Modelica.

4 Summary and Outlook

In this paper, the status of the TransiEnt Library after the completion of the ResiliEntEE project is presented. Due to the spatial expansion of the region of interest to Northern Germany, the models of the gas and power sector were extended.

In the power sector, the `ComplexPowerPort`, which integrates the complete complex bus voltage into the modeling, was introduced. This allows the modeling of interconnected electric power systems as they can be found on transmission grid level. In this context, existing models in the TransiEnt Library were revised according to this modeling approach and new models were added.

In the gas sector, the numerical efficiency of the concerning models was improved by simplifying the computation of the mass and energy balances. Furthermore, the pressure loss calculation is accelerated through various improvements. This leads to a strong reduction of numerical effort and enables gas grid calculation within the scope of Northern Germany.

To present the possibilities of the TransiEnt Library, an Integrated Energy System is discussed. With this system, it can be shown how a disturbance in the gas grid affects the integrated heat and power sectors, too. This dynamic simulation of disturbances and their consequences gives the foundation of a quantitative assessment of resilience. In this context, sensitivity analyses and the comparison with system cost and CO_2 emissions are possible.

With the completion of the project ResiliEntEE in June of 2021, all developed library extensions are freely available under the terms of the Modelica license agreement. Because of the high interest in the TransiEnt Library, a consortium was founded to promote further development. In this context, the projects IntegraNet and IntegraNet II already use the TransiEnt Library (Benthin et al. 2020). Therefore, the involved institutes Fraunhofer UMSICHT and Gas- and Wärme-Institut Essen e.V. as well as the XRG Simulation GmbH plan on contributing to the TransiEnt Library in the future.

The current version of the TransiEnt Library can be downloaded using the following link: https://www.tuhh.de/transient-ee/en/.

Acknowledgments

The authors greatly acknowledge the funding from the German Federal Ministry of Economic Affairs and Energy for the project "ResiliEntEE - Resilienz gekoppelter Energienetze mit hohem Anteil Erneuerbarer Energien" (ResiliEntEE - Resilience of integrated energy networks with a high share of renewable energies, project number: 03ET4048).

References

Agora Energiewende (2019). "European Energy Transition 2030: The Big Picture. Ten Priorities for the Next European Commission to Meet the EU's 2030 Targets and Accelerate Towards 2050". URL: https://www.agora-energiewende.de/en/publications/european-energy-transition-2030-the-big-picture/ (visited on 2021-02-15).

Andresen, Lisa, Pascal Dubucq, Ricardo Peniche Garcia, Günter Ackermann, Alfons Kather, and Gerhard Schmitz (2015). "Status of the TransiEnt Library: Transient Simulation of Coupled Energy Networks with High Share of Renewable Energy". In: *Proceedings Modelica Conference 2015*, edited by Modelica Association, 695–705. Linköping Electronic Conference Proceedings: Linköping University Electronic Press. DOI: 10.3384/ecp15118695.

Andresen, Lisa, Pascal Dubucq, Ricardo Peniche Garcia, Günter Ackermann, Alfons Kather, and Gerhard Schmitz (2017). "Transientes Verhalten gekoppelter Energienetze mit hohem Anteil Erneuerbarer Energien, Abschlussbericht des Verbundvorhabens: Laufzeit des Verbundvorhabens: 01.05.2013 bis 30.04.2017". DOI: 10.2314/GBV:1002659345.

Benthin, Jörn, Anne Hagemeier, Annika Heyer, Philipp Huismann, Joachim Krassowski, Christine Settgast, Ben Wortmann, and Klaus Görner (2020). "Gemeinsamer Abschlussbericht des Forschungsvorhabens Integrierte Betrachtung von Strom-, Gas- und Wärmesystemen zur modellbasierten Optimierung des Energieausgleichs- und

Transportbedarfs innerhalb der deutschen Energienetze". DOI: 10.13140/RG.2.2.17052.44166.

Bode, Carsten, Jan-Peter Heckel, Oliver Schülting, Anne Senkel, Christian Becker, Alfons Kather, and Gerhard Schmitz (2021). "Resilienz gekoppelter Energienetze mit hohem Anteil Erneuerbarer Energien: Abschlussbericht des Verbundvorhabens".

Bode, Carsten, and Gerhard Schmitz (2019). "Influence of Excess Power Utilization in Power-to-Heat Units on an Integrated Energy System with 100 % Renewables". In: *Proceedings of the 13th International Modelica Conference*, 413–22. Linköping Electronic Conference Proceedings: Linköping University Electronic Press. DOI: 10.3384/ecp19157413.

Cerbe, Günter, and Benno Lendt, eds. (2017). *Grundlagen der Gastechnik: Gasbeschaffung - Gasverteilung - Gasverwendung*. With the assistance of K. Brüggemann, M. Dehli and F. Gröschl. München: Hanser.

Dassault Systèmes (2021). *Dymola®*. Vélizy-Villacoublay, France. URL: https://www.3ds.com/products-services/catia/products/dymola/ (visited on 2021-03-24).

Hamburg University of Technology (2020). *TransiEnt Library*. Hamburg. URL: https://www.tuhh.de/transient-ee/en (visited on 2021-03-24).

Hamburg University of Technology, TLK-Thermo GmbH, and XRG Simulation GmbH (2021). *ClaRa*. Hamburg, Braunschweig. URL: https://claralib.com/index.php?lang=en (visited on 2021-03-24).

Heckel, Jan-Peter, and Christian Becker (2019). "Advanced Modeling of Electric Components in Integrated Energy Systems with the TransiEnt Library". In: *Proceedings of the 13th International Modelica Conference*, 759–68. Linköping Electronic Conference Proceedings: Linköping University Electronic Press. DOI: 10.3384/ecp19157759.

Huber, Marcia, Allan Harvey, Eric Lemmon, Gary Hardin, Ian Bell, and Mark McLinden (2018). "NIST Reference Fluid Thermodynamic and Transport Properties Database (REFPROP) Version 10 - SRD 23". DOI: 10.18434/T4/1502528.

Klimiuk, Ewa, Zygmunt Mariusz Gusiatin, Tomasz Pokój, and Sabina Rynkowska (2015). "ADM1-Based Modeling of Anaerobic Codigestion of Maize Silage and Cattle Manure – a Feedstock Characterisation for Model Implementation (Part I) / Modelowanie Kofermentacji Kiszonki Kukurydzy I Obornika Bydlęcego Za Pomocą ADM1 – Charakterystyka Wsadu Surowcowego (Część I)". *Archives of Environmental Protection* 41 (3): 11–19. DOI: 10.1515/aep-2015-0026.

Linkov, Igor, and José Manuel Palma-Oliveira, eds. (2017). *Resilience and Risk: Methods and Application in Environment, Cyber and Social Domains*. NATO science for peace and security series. Series C, Environmental security. Dordrecht: Springer Netherlands.

Modelica Association (2017-04). *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.

Modelica Association (2020). *Modelica® Standard Library*. Linköping, Sweden: Tech. rep. Linköping: Modelica Association.

Park, R. H. (1929). "Two-Reaction Theory of Synchronous Machines Generalized Method of Analysis-Part I". *Transactions of the American Institute of Electrical Engineers* 48 (3): 716–27. DOI: 10.1109/T-AIEE.1929.5055275.

Price, W. W., H. D. Chiang, H. K. Clark, C. Concordia, D. C. Lee, J. C. Hsu, S. Ihara et al. (1993). "Load Representation for Dynamic Performance Analysis (Of Power Systems)". *IEEE Trans. Power Syst.* 8 (2): 472–82. DOI: 10.1109/59.260837.

Senkel, Anne, Carsten Bode, and Gerhard Schmitz (2019). "Evaluating the Resilience of Energy Supply Systems at the Example of a Single Family Dwelling Heating System". In: *Proceedings of the 13th International Modelica Conference*, 655–62. Linköping: Linköping University Electronic Press. DOI: 10.3384/ecp19157655.

Senkel, Anne, Carsten Bode, and Gerhard Schmitz (2021). "Quantification of the Resilience of Integrated Energy Systems Using Dynamic Simulation". *Reliability Engineering & System Safety* 209:107447. DOI: 10.1016/j.ress.2021.107447.

TLK-Thermo GmbH, and Institut für Thermodynamik, Technische Universität Braunschweig (2021). "TILMedia Suite". URL: https://www.tlk-thermo.com/index.php/en/software/tilmedia-suite (visited on 2021-03-16).

Webster, John, and Carsten Bode (2019). "Implementation of a Non-Discretized Multiphysics PEM Electrolyzer Model in Modelica". In: *Proceedings of the 13th International Modelica Conference*, 833–40. Linköping Electronic Conference Proceedings: Linköping University Electronic Press. DOI: 10.3384/ecp19157833.

Wetter, Michael, Wangda Zuo, Thierry S. Nouidui, and Xiufeng Pang (2014). *Modelica Buildings Library*. URL: https://simulationresearch.lbl.gov/modelica/download.html (visited on 2021-03-24).

DLR Visualization 2 Library -
Real-Time Graphical Environments for Virtual Commissioning

Sebastian Kümper[1] Matthias Hellerer[1] Tobias Bellmann[1]

[1]Institute of System Dynamics and Control, German Aerospace Center (DLR),
{sebastian.kuemper, matthias.hellerer, tobias.bellmann}@dlr.de

Abstract

In this paper, the next generation of model-based visualization is introduced, the DLR Visualization 2 Library. This new real-time graphics environment for Modelica is equipped with a state of the art engine for physics based lighting calculation and high-definition render quality, simultaneous visualization of parallel running simulation models, new features like a modern streaming interface and a new, cleaner library structure. It enables the user to create graphical real-time environments for virtual commissioning of complex systems of systems and imaging based sensors. Some applications, as for example depth-camera data generation or rendering of point clouds or vectorized flow visualization are demonstrated in the use cases section of this paper.

Keywords: Visualization, Virtual Commissioning, Systems of Systems, Multi-Body

1 Introduction

The Modelica multi-body library allows for highly detailed simulations of mechanical structures. However, the integrated visualization for multi-body components available in all major Modelica tools contains only some simple geometrical forms, surfaces or static CAD models (Otter, Elmqvist, and Mattsson 2003). While this is sufficient for many engineering tasks, where only the structure and behavior of a system is of interest for the engineer, for virtual commissioning, a realistic graphical environment is sometimes necessary, especially if optical sensors should be used as part of an image processing pipeline (e. g. for homing a robot using camera data). Furthermore, more and more simulations, particularly those of "Systems of Systems", have to run in parallel to increase performance, creating the problem of asynchronous creation of model animation data to be processed by the render engine of the visualization tool.

To overcome these shortcomings, DLR developed the new DLR Visualization 2 library, an evolution and successor to the long available and continuously refined DLR Visualization Library from 2009 (Bellmann 2009; Hellerer, Bellmann, and Schlegel 2014).

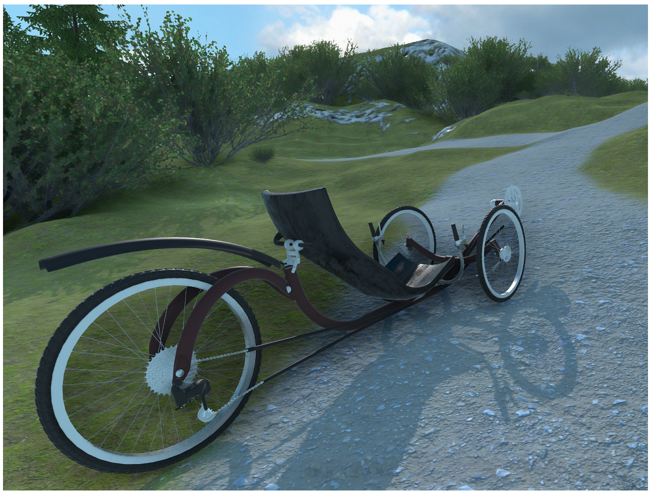


Figure 1. Recumbent bike visualized in the DLR Visualization 2 Library. Material properties of the CAD models as metalness or roughness are rendered in a physically meaningful way.

1.1 Modelica Visualization - State of the Art

A comprehensive review of alternative visualization methods for Modelica can be found in Hellerer, Bellmann, and Schlegel (2014). Since then some new work has been published, especially from Waurich and Weber (2017). Here, FMUs of the model are generated and integrated in the Unity engine within a newly developed Unity plugin. The FMU serves as data source for the visualizer elements (e. g. for their positioning) to enable high quality rendering with a state of the art engine. In Fuchs, Streblow, and Müller (2015), Python is used for the visualization of mass flows of thermal-fluid networks.

1.2 Virtual Commissioning

Within the life cycle of a product or plant, commissioning refers to the phase, where a new technical system is activated and used productively for the first time as a whole. However, in complex multi-component systems, this step oftentimes ends with problems, error messages and incompatibilities between the different systems and the desired task cannot be carried out as expected because of unforeseen differences between specification and reality. In order to avoid such problems and potential high costs caused by subsequent error solutions and eventual hardware/software redesigns, virtual commis-

DOI
10.3384/ecp21181197

sioning strives to commission the single system components before the completion/production of all other hardware components, by coupling them with a virtual environment, which simulates the yet unavailable components (VDI/VDE-Fachbereich Engineering und Betrieb 2016). Modelica is a useful tool here, as it enables the user to simulate yet to be produced or to be designed hardware components. One example would be the real-time connection of simulated hardware with the real world controller software/hardware of the system. However, if the controller relies on optical sensory input, for example as part of an image processing tool-chain, a real-time visualization with virtual sensors and camera streams becomes a necessity.

2 New Features of the DLR Visualization 2 Library

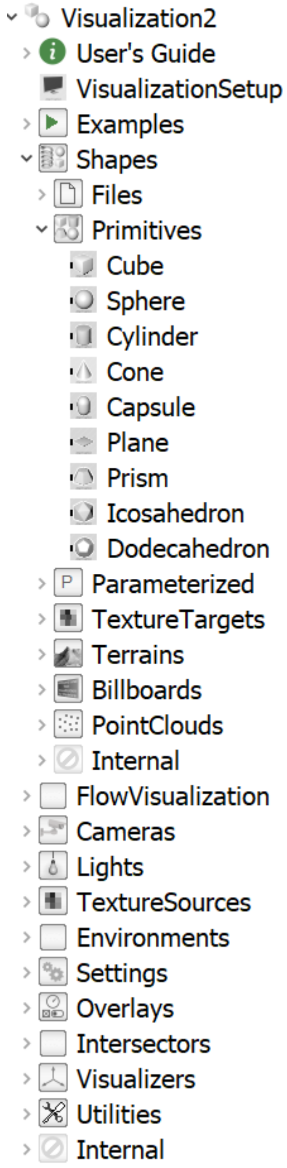
The DLR Visualization 2 Library is a completely new development. We integrated most of the features from the previous version and we also improved the interface and added new features. This improves the quality of created images and allows the library to be used for more use cases which are also useful for virtual commissioning. The structure of the new library can be seen in Figure 2.

2.1 Improved Rendering Quality

The DLR Visualization 2 Library uses a completely new and modern rendering backend which is based on the real-time 3D engine Unigine (Unigine 2021). This allows for a greatly increased rendering quality. An example image can be seen in Figure 1.

One core change is the material rendering. Previously, the materials were parameterized by abstract values that were not based on real world properties. In the new version the parameters are based on the physical appearance of real world objects and are consequently more intuitive to the user (Greenberg et al. 1997). Every object has three properties: color, metalness and roughness. The color defines the dominant color of the object. Metalness is a boolean value which defines whether the object is made of metal or not. Roughness is a value between 0 and 1 which defines how rough the rendered object should be. A value of 0 indicates that the object is perfectly polished and light gets perfectly reflected. A value of 1 indicates that the object is very rough in a microscopic sense. An example for varying material properties can be seen in Figure 3. These properties can either be set for simple objects, overridden for file objects or directly imported from a gltf file (Khronos 2021). Other CAD-File types are supported but the resulting material may not be the intended material.

Another big improvement is the scene lighting. The light sources have a more realistic look and they produce dynamic high quality shadows. The shadows can also be disabled per light source or per object to give the user more control over the result. The brightness is set via a physical value in lux. The lighting also includes ambient occlusion which enhances cavities by simulating small distance shadows.



- Visualization2
  - User's Guide
  - VisualizationSetup
  - Examples
  - Shapes
    - Files
    - Primitives
      - Cube
      - Sphere
      - Cylinder
      - Cone
      - Capsule
      - Plane
      - Prism
      - Icosahedron
      - Dodecahedron
    - Parameterized
    - TextureTargets
    - Terrains
    - Billboards
    - PointClouds
    - Internal
  - FlowVisualization
  - Cameras
  - Lights
  - TextureSources
  - Environments
  - Settings
  - Overlays
  - Intersectors
  - Visualizers
  - Utilities
  - Internal

Figure 2. The new Visualization 2 library structure with separate primitives for better visibility on modeling level

2.2 Depth Rendering and Point Clouds

In the field of automation and robotics, sensor systems that create depth information or spatial data are common. Stereo cameras may create depth images, i. e. images where each pixel represents the distance to the viewed object. Laser scanners or sonars may create a list of points that approximate the scanned objects. We added several options to visualize this data and also added the virtual camera `DepthCamera`, to create depth image data in a virtual environment, so that these types of sensors can be simulated.

It is possible to directly visualize the depth field created by the `DepthCamera` within the visualization window. The distance of the camera to the viewed object is then encoded in the color of the camera image pixels. For an example, see the use case in section subsection 3.1. For a better visualization of small distance

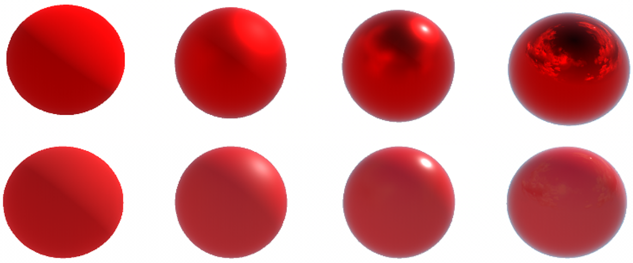


Figure 3. Example of material properties of a sphere. Top row: metalness set to true, Bottom row: metalness set to false. Decreasing roughness from left to right.

differences, the user can enable Eye-Dome-Lighting (Ribes and Boucheny 2011) by applying a dummy shading technique. The data can also be sent to other applications as described in subsection 2.3.

Depth image data can also be visualized in 3D with `DepthMesh`. This is a plane with a given resolution that is deformed according to the depth image. Together with the position and orientation of the camera, the object viewed by the camera can be accurately reconstructed in 3D.

A more versatile solution are point clouds. Here individual points are visualized via small squares. A point cloud can also be generated from a `DepthCamera` with the `CameraPointCloud`. This delivers nearly the same result as the `DepthMesh`, but each pixel of the depth image will correspond to an individual point instead of a closed mesh.

In addition to `CameraPointCloud`, there exists `StreamPointCloud` where the points are received from a network source, `RawPointCloud` where the points can be set directly through Modelica and `FilePointCloud` where the points are loaded from a .xyz-file. The data of multiple time points of point clouds can be combined into a grid, so that the complete scene can be viewed instead of a single time point. The user also has the possibility to filter the data by the quality data from the sensor and color the point clouds accordingly.

An example application can be seen in Figure 4. Here a remote operated underwater vehicle (ROV) is simulated. It has a panning depth camera attached to its front to view the surroundings. On the top left image, the ROV with the landscape is shown. The current view of the depth camera is shown with a `DepthMesh` in red. On the top right is the colored depth image of the camera with enabled eye-dome-lighting. On the bottom left is a `StreamPointCloud` which receives the points from the camera via network. On the bottom right is another `StreamPointCloud`, which combines the information from previous time steps to visualize the complete scanned path. It is also possible to replace the `DepthCamera` source with a real world sensor so that real sensor data is visualized instead.

2.3 Improved Streaming Support

Some simulations, particularly in virtual commissioning scenarios, are controlled by external applications that rely on data from processed camera images. In order to provide such synthetic camera data for external image processing pipelines, we added support to stream virtual camera images to arbitrary targets. An example process can be seen in Figure 5 where the simulation is controlled from the output of an image processing software. The visualization of the simulation creates a depth image, which is sent to an external image processing pipeline to create control inputs for the simulation.

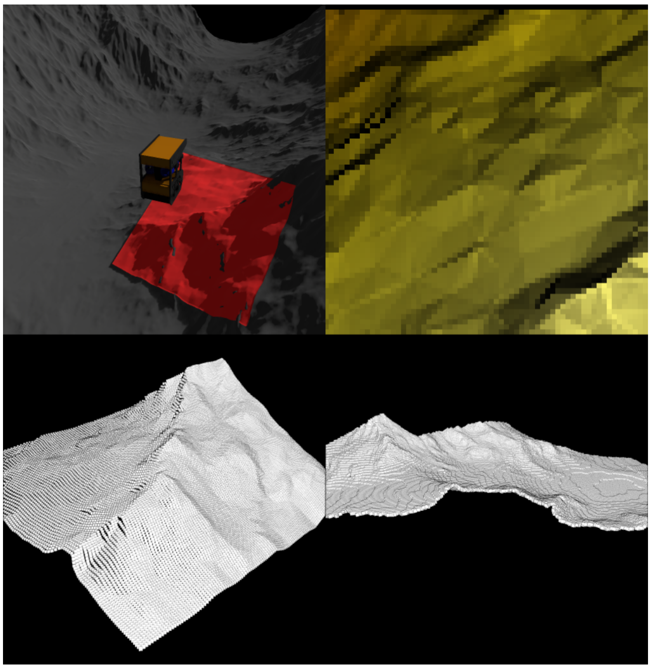


Figure 4. Example application for the use of depth information. Top left: ROV with depth camera floats above terrain, depth mesh in red is viewed area. Top right: current view with wrong colors and shading. Bottom left: current reconstructed landscape based on depth image. Bottom right: point cloud of combined time steps.

Every virtual camera has the possibility to stream its current view to a network target. The resolution of the stream is defined by a parameter that is independent from the resolution of the screen. The user can also set the target frame-rate of the stream to either reduce the workload of the visualization or to simulate real world limitations. The supported network protocols are UDP, TCP and RTSP.

The new `DepthCamera` can also stream their depth information to network targets, to simulate the results of a stereo vision pipeline. The depth information can be streamed using different options: Either the depth information is streamed via an encoded video (Pece, Kautz, and Weyrich 2011) or the data depth information of each pixel is packed into a FlatBuffers (Google 2014) package (either as 2D image or as 3D data) and subsequently streamed.

2.4 Testing Abilities

During testing of models, engineers want to change multiple parameters and compare the results of several simulations. In Modelica, it is feasible to vary parameters automatically (e. g. using Monte Carlo methods of the Optimization Library (Joos et al. 2002)), so the automated creation of comparison images should also be possible. This is why we added support to control the creation of screenshots and videos from Modelica.

To create screenshots automatically, the user specifies the simulation time points at which a screenshot should be taken. During simulation, the visualization will halt for a brief moment at the requested time points and create a screenshot. This also works for extremely fast simulations. The simulation time is appended to the given path so that the user can make several screenshots during one simulation. In Figure 6 (Pignède and

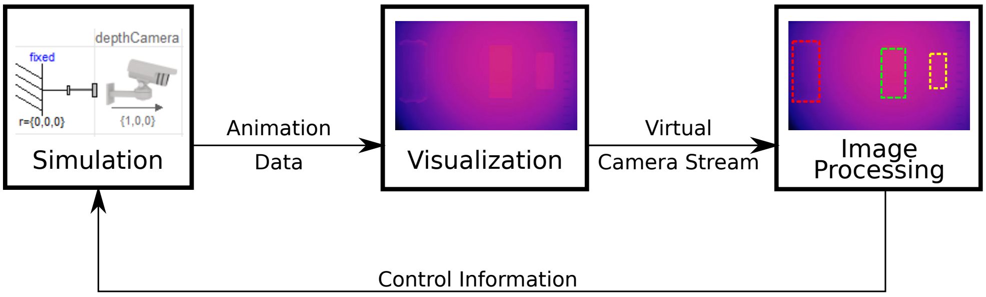


Figure 5. The process for a simulation controlled by cameras. The simulation sends animation data to the visualization, which renders a depth image and streams it to an external application. This application processes the depth image, calculates control information and sends it back to the simulation.

Lichtenheldt 2022) several screenshots are automatically created at different time steps for a short overview of the simulation.

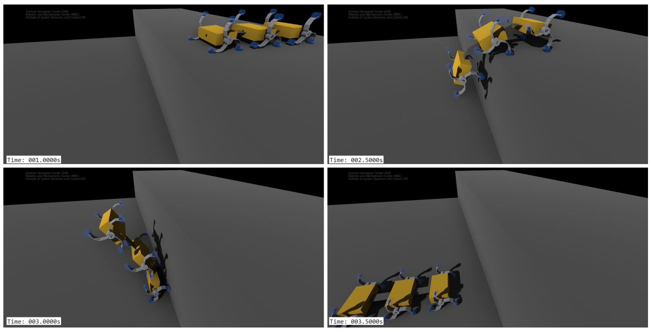


Figure 6. Screenshots automatically created at different time steps

Normally, a screenshot of all cameras that are displayed in the main window is created. A screenshot from a single camera can be triggered inside the individual cameras. Here, the screenshots can be taken at pre-defined times or when certain simulation events are happening. This is not only useful for checking results, but can also be used to mimic the functionality of a real world camera that is only able to create still images instead of videos.

Another option to compare results are auto-generated videos. When this option is enabled, the visualization will automatically create a video with the specified settings and save it to the specified path. This allows the user to run a multitude of simulations with varying parameters and simply compare the resulting videos.

2.5 Flow Visualization

In the previous version of the visualization library it was already possible to visualize the flow of media inside a pipe. In the new version the path is defined by a series of points. At each point, the user can define the desired position, the speed and the color of the visualized flow elements. The path in between the points is either interpolated cubically or linearly. The flow elements themselves can either be visualized with cones, rings, arrows or even imported CAD-Files. When using CAD-Files, the user can control the up-vector so that rotations along the flow axis are possible. This can also be used to visualize objects that consist of multiple small moving objects. In Figure 7 this is used to visualize the chain of a bicycle.

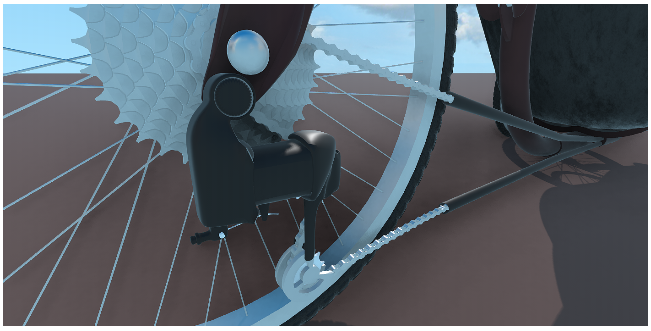


Figure 7. A moving chain is visualized with the `PathVectorFlow`

Another new addition is the visualization of flows inside of a three dimensional field. This can be used to visualize the airflow around objects, like cars or wings. The vector field consists of a three dimensional array of points. At each point the user can specify a vector and its color. There exist two methods to visualize vector fields, `GridVectorField` and `GridVectorFlow`. `GridVectorField` visualizes the individual vectors as arrows which have the specified direction, length and color. `GridVectorFlow` visualizes the flow inside of this field. In order to do this, the user defines seeding points. At these points particles are inserted that follow the flow inside the field. The flow can also be visualized by lines that can either follow a virtual particle over time through the field or they can be simulated in a single time step. The lines are colored with the colors defined in the grid. An example flow around a sphere can be seen in Figure 8.

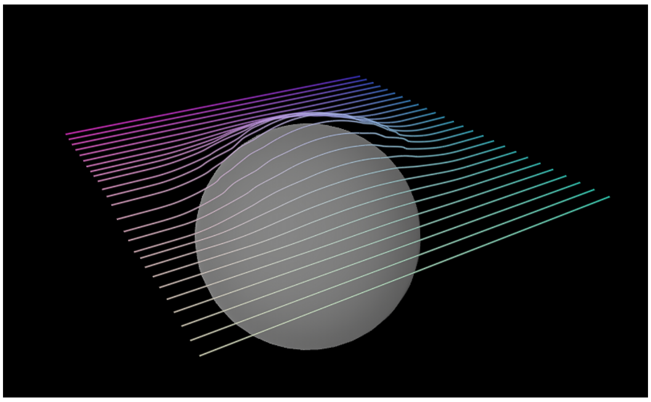


Figure 8. Visualized air flow around a sphere with the new `GridVectorFlow`

2.6 CAD Array

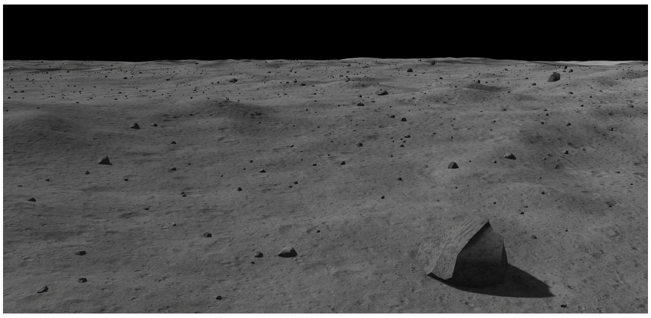


Figure 9. Use of the CAD array for the visualization of a distribution of 1.7 million rocks on a simulated moon landscape

There are situations where one would need to visualize large amounts of similar, static objects, e. g. rocks on a surface. Doing this with the normal `CADFile` leads to many equations in Modelica and therefore slows down the initialization and simulation significantly.

To counter this, `CADArray` allows the user to visualize a large number of instances of the same CAD file. For each instance the position, rotation and scale can be specified. This can either be done directly in Modelica via arrays or a file can be loaded that holds all of the information.

An example application can be seen in Figure 9. A lunar landscape had to be enhanced with rocks to provide additional detail and obstacles for a lunar landing simulation. The visualized moon is fairly large, so in order to achieve the desired obstacle density around 1.7 million rocks have to be placed and visualized on the surface. This would neither be possible to simulate nor to visualize without `CADArray`.

2.7 Level of Detail

Visualizing large amounts of objects with many details can be very challenging to render, so we introduced the possibility to specify multiple levels of detail (LoD).

A version of the object with many details is chosen as the main file. Additionally multiple LoDs can be added. Each of these LoDs is defined by a CAD file and a minimum visibility distance. At this distance the object will be visualized by the LoD's CAD file and the CAD file of the lower levels will be hidden.

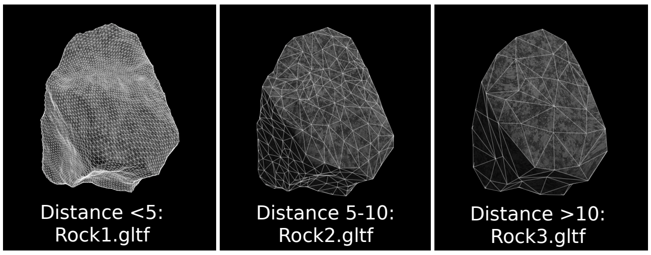


Figure 10. A rock displayed with varying polygon count at different distances. The user can specify what CAD files should be shown at which distance to reduce the stress on the GPU, while keeping visual quality high.

An example of a rock with multiple LoDs can be seen in Figure 10. When the viewer is less then five units away, the very detailed Rock1.gltf is shown. When the viewer is farther away at a distance of between 5 and 10 units, the less detailed Rock2.gltf is shown. At a distance over 10, the even less detailed Rock3.gltf is shown. This reduces the load on the GPU, while keeping visual quality high.

2.8 Additional Improvements

Interface

We completely reworked the Modelica interface in the DLR Visualization 2 Library to improve the overall usability of the library. We separated the primitives into individual objects so that at a glance, the user can see what kind of objects are displayed. Additionally, primitives and other objects with a main color are displayed in that color in the Modelica interface. A comparison can be seen in Figure 11.

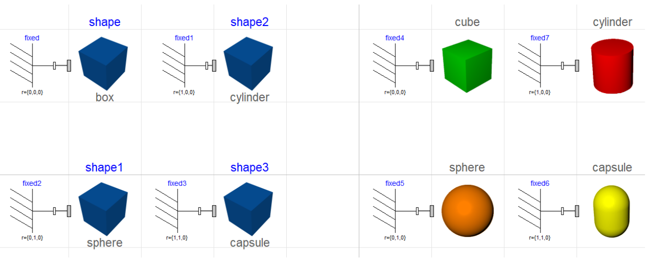


Figure 11. Comparison of the Modelica blocks for Primitives from the old library (left) to the new library (right). In the new library the primitives are split into different objects and the color is shown in the model

Rigged CAD File

We added the possibility to manipulate objects with bones that are defined in the CAD file. Manipulating objects with virtual bones is a standard in computer graphics. The most prominent use case is the creation of animated humans or animals, as it allows for the deformation of the skin. However, bones can also be used for mechanical structures like robots or rovers. Here, the advantage over individual subobjects is that the relative position of the objects is given in the CAD file and the user does not have to manually position them in the simulation model. For easier use, we added a Modelica function which extracts the bone information from the specified CAD file to create a model where the bones can be identified by their given names.

VR camera

We improved the usage of VR-Headsets. Now, every headset that uses the OpenVR standard is supported. Optionally, overlay items displayed for the VR user on a plane, which floats in front of the user, are now possible.

Another feature is the addition of green screen support (chroma keying). Here, a camera is mounted on the VR headset. The camera image will be displayed in the VR image and combined with the virtual image, which will be displayed at the masked green areas. This can be used to merge a real environment with a virtual environment, e. g. a real cockpit with a virtual landscape.

Window setup from within Modelica

The visualization window arrangement can now be controlled from within Modelica. This may be used for the automatic setup of a multi screen simulation. This includes the position, size and style. The position and size is specified relative to the screen size, so that the appearance is independent from the screen resolution. The window can be a normal window with title bar and border, without any borders and title or fullscreen. It is also possible to create additional windows with the same properties.

Camera Position Back-Channel

The user can retrieve the position and orientation of every camera in Modelica. The cameras have an optional output for the position vector and the orientation matrix. These can be used to construct a frame and thereby objects can be attached to a user controlled camera. This also works for the `VRCamera`. This means that objects can be placed relative in the view of the VR user to, e. g. to create some additional interface.

Textures

In the previous version of the Visualization Library, textures (i. e. images or videos) were treated differently depending on the source. In the DLR Visualization 2 Library there is a common interface: the texture buffer. All individual sources (virtual camera, network stream, webcam, file) are treated the same and are interchangeable. This allows the user to iteratively test the simulation by simply exchanging the source of a texture and nothing else.

Overlay Positioning

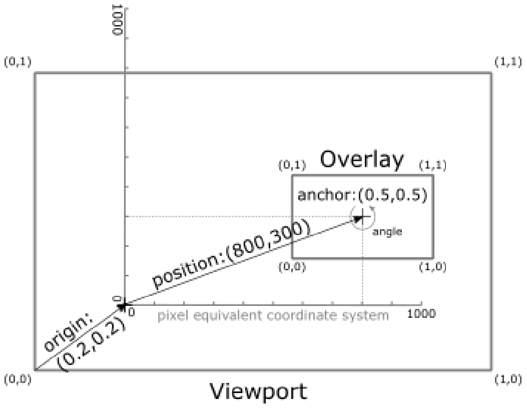


Figure 12. Possibilities of the overlay positioning mechanism. The origin is given relative to the viewport. The overlay position is relative to the origin and is given in pixel equivalents to allow for exact positioning of multiple interacting items. Finally the anchor can be used to change the handle position of the item.

We added more possibilities for the positioning of overlay items (formerly HUDs). In the previous version, the positioning was rather limited. The origin was always on the bottom left of the viewport. In Figure 12 the new possibilities can be seen. This is controlled by three positioning parameters:

- origin: relative position in the viewport
- position: relative to origin in pixel equivalents (equals 1/1000th of a defined side of the viewport)
- anchor: handle position of item relative to item size

This allows the user to position the overlay items to be aligned to any part of the viewport and also position it by any part of the item while still keeping the possibility for items to be positioned exactly for them to interact with each other.

Environments

Environments define the overall look of the simulation. They provide options for the sun light, the ambient light and the background. For the background, either a skybox can be defined or the sky color is calculated based on the position of the sun. Haze can be added to provide a more realistic visualization of large scale earth-based simulations.

3 Use Cases

While still under development, the DLR Visualization 2 Library is already used for many internal projects. A few practical applications will be introduced here.

3.1 Virtual Commissioning of a Robot Cell

Visualizing the overall assembly of a system early in the development process and during virtual commissioning has many benefits. Engineers can, for example, immediately asses the size of complex work-spaces with many moving parts to avoid collisions. Further, it is often times necessary to present a machine or plant to stakeholders long before it is physically built. However, if real components rely on optical sensor values, e. g. an imaging pipeline to control a pick and place task, this can be addressed in the simulation visualization as well.

Such an application is shown in Figure 13: Two interacting robots assemble housings for network components (Bellmann, Seefried, and Thiele 2020; Reiser 2021). For this task they are equipped with depth cameras on the side of their tools to detect the exact position of a workpiece. In this complex application the visualization is part of virtual commissioning. It creates depth images as shown in Figure 14 and streams them to an image processing algorithm. Its results then provide positioning input to the robot controllers and thereby close the control loop over visualization and image processing.

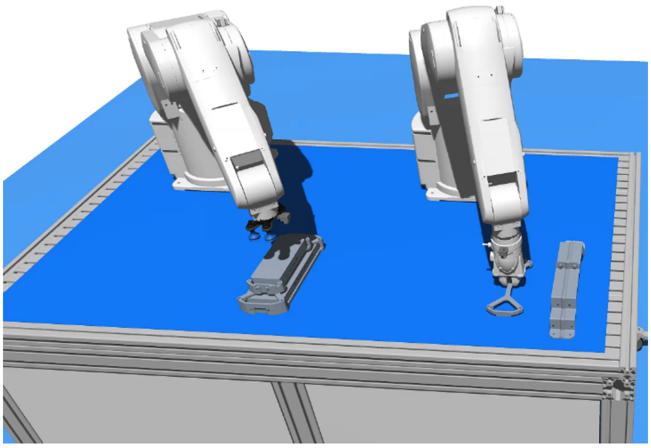


Figure 13. Example for virtual commissioning of a robot cell. Two cooperative robots assemble housings for network components.

3.2 Visualization of Parallel Running Simulations

The Helmholtz Future Project ARCHES develops teams of heterogeneous robotic agents for deep-sea and extraterrestrial exploration with the goal of improving efficiency and robustness

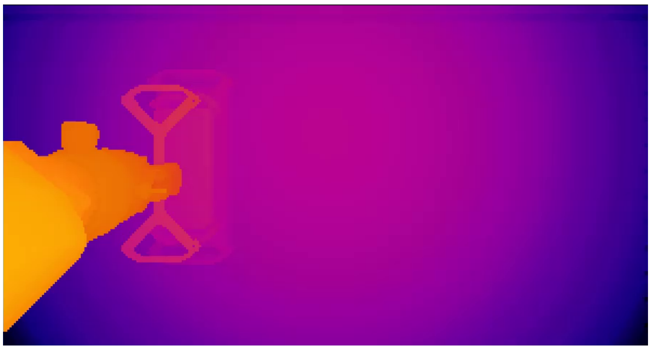


Figure 14. Depth image produced by a stereo camera, mounted on one of the robots tool. Color denotes distance from camera.

under extreme conditions (Schuster et al. 2020). During the development of such a team, only a limited number of robots is actually available and using those for regular software tests or during development is complex and laborious. So simulations play an important role during development of robot teams. Yet the simulation of a large number of agents is again difficult, especially in an efficient manner. Almost all available Modelica implementations are very limited when it comes to parallelization. Typically the whole simulation is run in one monolithic block on a single CPU core (Gebremedhin 2019). So to simulate a whole team of robots, each agent is split into a single simulation. The simulations are then only loosely coupled, but they all have to be visible in one common visualization. The DLR Visualization 2 library supports the rendering of multiple simulations in one visualization container.

In this use case a multi-agent simulation has been created to simulate multiple rovers on a virtual but realistic landscape. It provides a network API which allows the developers of control algorithms for the rovers to send commands to the robots and to receive status data from them. Among this data are video streams of the simulated robots cameras e. g. for a navigation pipeline. Figure 15 shows a common visualization of multiple largely independent, simulations in one visualization container.

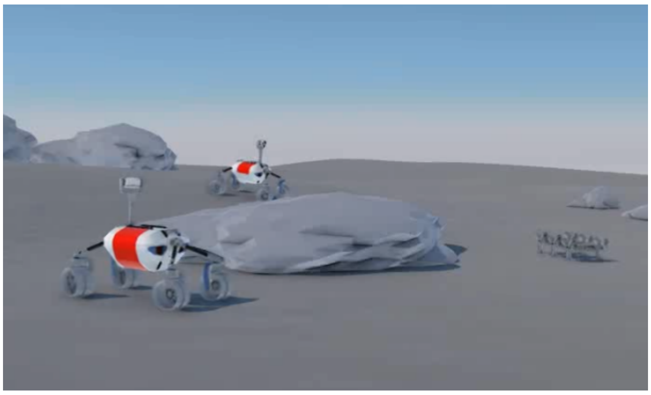


Figure 15. Multiple rovers, each from a separate simulation, rendered in one visualization

The virtual camera images should be as close to real ones as possible to give operators a realistic impression during development and training. The video streams might even be used as input for image processing algorithms for tests and during development as presented in Wedler et al. (2017).

3.3 Automated Testing of Rover Operations

The Martian Moon eXploration mission (MMX) is a cooperative effort by the Japanese space agency JAXA, the French space agency CNES, and the German space agency DLR. Together they plan to explore the martian moons Phobos and Deimos (Ulamec et al. 2019). An important part of this project is the deployment of a mobile rover on Phobos (Bertrand et al. 2019; Buse et al. 2021). Designing the first wheeled rover for use on Phobos is a highly complex task. Little is known about many environmental factors such as the surface structure and what is known, like the micro gravity, poses a number of new problems that no previous wheeled exploration rover has ever faced (Bertrand et al. 2019; JAXA 2017; Lange 2020).

An environment like this cannot be recreated anywhere on earth, therefore simulations and their visualization play a central role during the development. One of the most challenging tasks is the simulation of the wheel-ground contact under low-g conditions. Without a visualization it is very hard to get a real insight into the complex interaction of multiple contact points in a non-intuitive environment.

Such visualizations are also directly involved in the development of the locomotion planning tool for the rover. A simulated navigation and wheel cameras will be used to determine the feasibility of a planned locomotion trajectory.

Finally, space missions also have to present their work and their results to the general public in an interesting and engaging fashion. For this, nowadays, high quality render images of the mission are generally expected. Figure 16 shows such an image, presented, for example, on JAXAs MMX mission website (JAXA 2020).

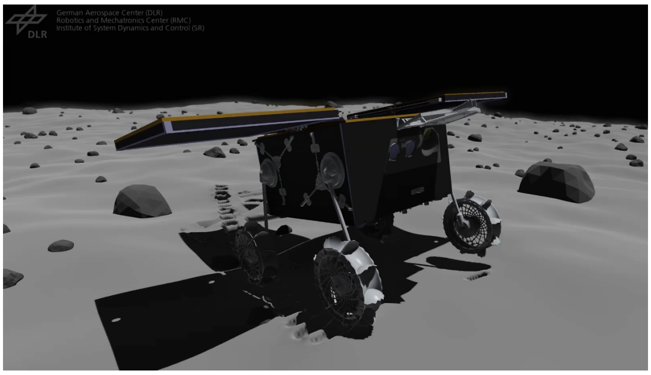

DLR
German Aerospace Center (DLR)
Robotics and Mechatronics Center (RMC)
Institute of System Dynamics and Control (SR)

Figure 16. Visualization of the simulated MMX Rover on Phobos. The rover moves on soft soil with large rocks in a micro gravity environment.

4 Conclusion and Outlook

Whilst the possibility to export FMU of Modelica models enables virtual commissioning of the physical system behavior, the usage of visually rendered environments for virtual commissioning is not standardized yet. Especially the integration of real imaging pipelines requires a high-definition real-time rendering engine with video stream capabilities, not available in most Modelica tools. In this work we presented the commercially available DLR Visualization 2 Library and its new features aiming at integrating such pipelines. Another big step in usability is the possibility to visualize the results of multiple, parallel running simulations.

For future developments, it is planned to extend the DLR Visualization 2 Library with a generic C++ API, a Matlab/Simulink interface and a JULIA library. The library, interfaces and the render tool itself will be provided in a free community edition limited to real-time multi-body visualizations and a commercial version containing all features.

Acknowledgments

The authors would like to thank all colleagues at DLR helping during the implementation and testing of the library, especially Robert Reiser, Fabian Buse, Antoine Pignede and Miguel Neves, who also provided some of the examples and use cases. Additionally, the authors would like to thank GEOMAR for the model of the ROV. This work was partially funded by the HVF 68 Project Lighthouse.

References

Bellmann, Tobias (2009). “Interactive simulations and advanced visualization with modelica”. In: *Proceedings of the 7th international Modelica conference*. Linköping University Electronic Press.

Bellmann, Tobias, Andreas Seefried, and Bernhard Thiele (2020-10). “The DLR Robots library – Using replaceable packages to simulate various serial robots”. In: *Proceedings of Asian Modelica Conference 2020* (Tokyo, Japan). Linköping University Electronic Press. DOI: 10.3384/ecp2020174153.

Bertrand, Jean et al. (2019-05). “Roving on Phobos: Challenges of the MMX Rover for Space Robotics”. In: *15th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)* (Noordwijk, Netherlands). ESA/ESTEC.

Buse, Fabian et al. (2021-10). “Wheeled locomotion in milligravity: A technology experiment for the MMX Rover”. In: *72th International Astronautical Congress* (Dubai, UAE). The International Astronautical Federation.

Fuchs, Marcus, Rita Streblow, and Dirk Müller (2015). “Visualizing simulation results from modelica fluid models using graph drawing in python”. In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. 118. Linköping University Electronic Press, pp. 737–745.

Gebremedhin, Mahder (2019-01). *Automatic and Explicit Parallelization Approaches for Equation Based Mathematical Modeling and Simulation*. Linköping University Electronic Press. DOI: 10.3384/diss.diva-152789.

Google (2014). *FlatBuffers*. URL: https://google.github.io/flatbuffers (visited on 2021-04-22).

Greenberg, Donald P. et al. (1997). “A Framework for Realistic Image Synthesis”. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’97. USA: ACM Press/Addison-Wesley Publishing Co., pp. 477–494. ISBN: 0897918967. DOI: 10.1145/258734.258914. URL: https://doi.org/10.1145/258734.258914.

Hellerer, Matthias, Tobias Bellmann, and Florian Schlegel (2014). “The DLR Visualization Library-recent development and applications”. In: *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*. 096. Linköping University Electronic Press, pp. 899–911.

JAXA (2017-08). *Gravity both too strong and too weak: landing on the Martian moons*. URL: https://mmx-news.isas.jaxa.jp/?p=331&lang=en (visited on 2021-04-22).

JAXA (2020-10). *The MMX Rover is undergoing tests for landing*. URL: https://mmx-news.isas.jaxa.jp/?p=1271&lang=en (visited on 2021-04-22).

Joos, Hans-Dieter et al. (2002). “A multi-objective optimisation-based software environment for control systems design”. In: *IEEE International Conference on Control Applications and International Symposium on Computer Aided Control Systems Design, 2002-09-18 - 2002-09-20, Glasgow, Scotland (UK)*. IEEE, pp. 7–14.

Khronos (2021). *glTF Specification Webpage*. URL: https://www.khronos.org/gltf (visited on 2021-04-22).

Lange, Michael (2020-09). *First tests for landing the Martian Moons eXploration Rover*. URL: https://www.dlr.de/content/en/articles/news/2020/03/20200930_in-free-fall-to-the-martian-moon-phobos.html (visited on 2021-04-22).

Otter, Martin, Hilding Elmqvist, and Sven Mattsson (2003-11). “The New Modelica MultiBody Library”. In: pp. 311–330.

Pece, Fabrizio, Jan Kautz, and Tim Weyrich (2011). “Adapting Standard Video Codecs for Depth Streaming”. In: *Joint Virtual Reality Conference of EGVE - EuroVR*. Ed. by Sabine Coquillart, Anthony Steed, and Greg Welch. The Eurographics Association. ISBN: 978-3-905674-33-0. DOI: 10.2312/EGVE/JVRC11/059-066.

Pignède, Antoine and Roy Lichtenheldt (2022). “Modeling, Simulation and Optimization of the DLR Scout Rover to Enable Extraterrestrial Cave Exploration”. In: *The 6th Joint International Conference on Multibody System Dynamics (IMSD) and The 10th Asian Conference on Multibody Dynamics (ACMD), New Delhi, India: October 16-20, 2022*. Accepted for publication.

Reiser, Robert (2021). “Object Manipulation and assembly in Modelica”. In: *Proceedings of the 14th international Modelica conference*. Linköping University Electronic Press.

Ribes, Alejandro and Christian Boucheny (2011-04). “Eye-Dome Lighting: a non-photorealistic shading technique”. In: URL: https://blog.kitware.com/eye-dome-lighting-a-non-photorealistic-shading-technique (visited on 2021-04-22).

Schuster, Martin J. et al. (2020-10). “The ARCHES Space-Analogue Demonstration Mission: Towards Heterogeneous Teams of Autonomous Robots for Collaborative Scientific Sampling in Planetary Exploration”. In: *IEEE Robotics and Automation Letters* 5.4, pp. 5315–5322. DOI: 10.1109/lra.2020.3007468.

Ulamec, S. et al. (2019-10). “A rover for the JAXA MMX Mission to Phobos”. In: *70th International Astronautical Congress* (Washington DC, USA). The International Astronautical Federation. Chap. A3.

Unigine (2021). URL: https://unigine.com/ (visited on 2021-04-22).

VDI/VDE-Fachbereich Engineering und Betrieb (2016-08). *Virtual commissioning - Model types and glossary*. Tech. rep. VDI/VDE 3693. Verein Deutscher Ingenieure e.V., p. 35.

Waurich, Volker and Jürgen Weber (2017). “Interactive FMU-based visualization for an early design experience”. In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. 132. Linköping University Electronic Press, pp. 879–885.

Wedler, Armin et al. (2017-09). “First Results of the ROBEX Analogue Mission Campaign: Robotic Deployment of Seismic Networks for Future Lunar Missions”. In: *68th International Astronautical Congress* (Adelaide, Australia). The International Astronautical Federation.

Towards a Modelica OPC UA Library for Industrial Automation

Bernhard Thiele[1]

[1]Institute of System Dynamics and Control, German Aerospace Center (DLR), Germany,
bernhard.thiele@dlr.de

Abstract

Open Platform Communications Unified Architecture (OPC UA) is often named as a prospective enabler for future automation systems integrations, as for example envisioned in the German Initiative Industrie 4.0. The DLR OPC UA Modelica library connects OPC UA with the Modelica world. There are two main goals: First, OPC UA server capabilities for emulating the communication interface of (physical) hardware components in order to create component simulations, *e.g.*, for virtual commissioning. Second, OPC UA client capabilities for interacting with real-world hardware components, *e.g.*, for process visualization and monitoring or interactive simulation and control purposes. The library works on Windows and Linux platforms. It is tested using the Modelica environments Dymola and OpenModelica.

Keywords: OPC UA, Industry 4.0, Robotics, Modelica

1 Introduction

Open Platform Communications Unified Architecture (OPC UA) is often named as a prospective enabler for future automation systems integrations. It is considered as an existing technology which can cover various communication aspects for flexible production lines as envisioned in the German Initiative Industrie 4.0 and is also prominently mentioned in the Reference Architecture Model Industrie 4.0 (ZVEI 2015).

The significance of OPC UA as a core communication protocol for future automation systems has prompted the development of a Modelica library with the goals of enabling:

- Component simulations: The simulated component has an *OPC UA server* instance which mimics the *OPC UA server* of an actual system component down to some level of acceptable fidelity. Modelica is used for modeling the component's behavior in order to provide realistic values for the simulated process variables. Consumers of the data are OPC UA clients, *e.g.*, in supervisory control and data acquisition (SCADA) sytems. Possible scenarios are Hardware-in-the-Loop (HIL) simulation or virtual commissioning.

- Interactive simulation and control: The application has an *OPC UA client* instance which interacts with physical hardware components. The Modelica-based application uses actual hardware process variables which it queries from an OPC UA server instance on the real hardware component. Possible scenarios include querying quantities from hardware components for process visualization and monitoring and (model-based) high-level control tasks.

The term *component simulation* is used in the sense as defined in (Harrison and Proctor 2015), where emulation is defined as "the production of artificially created signals to represent the physical presence of some part of the manufacturing process" and a simulation of one component with emulation capabilities is termed as a component simulation.

The following paragraphs briefly describe existing approaches of supporting OPC UA connectivity in Modelica environments.

The OpenModelica environment (Fritzson et al. 2020) supports an option for starting an embedded OPC UA server which maps the simulation variables into an address space which can be monitored by OPC UA clients. In addition, a connected client can control the progress of the simulation by setting specific simulation control variables through the OPC UA interface. The feature was added during the OpenCPS project and is briefly described in its deliverable report (Sjölund and Asghar 2018). It is worth noting that the server implementation is at the tool level, *i.e.*, OpenModelica specific. Additionaly, while the goal of our library-based OPC UA server approach is to model the communication interface of (physical) hardware components (in order to create component simulations for HIL simulation and virtual commissioning), the goal of the OpenModelica tool-based OPC UA server is to facilitate debugging and monitoring of (embedded) OpenModelica real-time simulations.

The Modelica OPC UA libraries from Wolfram (Wolfram Research 2021) and ESI (ESI Group 2021) provide OPC UA client functionality. Their goals regarding the Modelica OPC UA client interface are similar to the goals of our own library. The significant difference is that in their present state OPC UA server capabilities are not in the scope of these libraries.

One common trait of all the approaches for connecting OPC UA to Modelica (including our own), is that they rely on the open62541 open-source implementation of OPC UA (*open62541* 2021) as underlying technology stack.

DOI
10.3384/ecp21181205

2 OPC UA

Besides being a hardware-independent communication protocol, the interesting capability of OPC UA is the ability of semantic information modeling. This information modeling allows an object-oriented style of modeling devices including hierarchical composition, object types (≈ classes), type hierarchies (≈ inheritance), instantiation, and (customizable) relations between objects. Indeed, if desired, there is the option of using the well-established Unified Modeling Language (UML) as base for OPC UA information model design as described by Pauker et al. (2016).

The base elements of OPC UA's meta model are nodes. These nodes are connected by typed references resulting in an undirected graph forming the *OPC UA Address Space*. Several node classes are predefined by the standard. Each node has a set of *attributes* which depend on the node class. They can be mandatory or optional. An attribute which is mandatory for any node is its *NodeId* for uniquely identifying the node.

The information model can be extended by so-called *companion specifications*. Simply speaking, one could compare those to libraries in conventional programming languages, *e.g.*, they usually define new object types which can be instantiated. Anyone, *e.g.*, device manufactures, can define own extensions. However, companion specifications particularly facilitate domain specific standardization.

A lot of work in our group is centered around robotic applications. Hence, integrating the OPC UA Companion Specification for Robotics (OPC 40010-1 2019) is of particular interest. The corresponding specification work is driven by the VDMA Robotics Initiative with the goal of specifying an OPC UA information model for complete motion device systems (including, but not limited to, conventional industrial robots), split up into several parts (Part 1 to Part n). At the time of this writing, the group has so far completed and released Part 1. It provides a basic description of a motion device system with the aim of pushing condition data vertically into higher level manufacturing systems. Future extension will cover further use cases, *e.g.*, to configure and control a robot. An example exploring interesting possibilities is given by Profanter et al. (2019) who propose an extension which provides a standardized (hardware-agnostic) control interface for robot manipulators.

3 Overview

Figure 1 gives an overview over the library structure and shows a basic server example.

The package browser at the left side of Figure 1 shows an `OPCUAServer` and `OPCUAClient` block which can be dragged and dropped into the diagram layer. These are the central blocks in the library for creating an OPC UA server or client, respectively. The `LeanLoggerInit` block ensures that messages within the external C code are forwarded to the Modelica environment and also registers the Modelica environment's implementation of the `ModelicaAllocateStringWithErrorReturn()` to the interfaced dynamic link library (DLL, Windows), or shared object library (SO, Linux). The `Functions` package contains the function interface to the external C code.

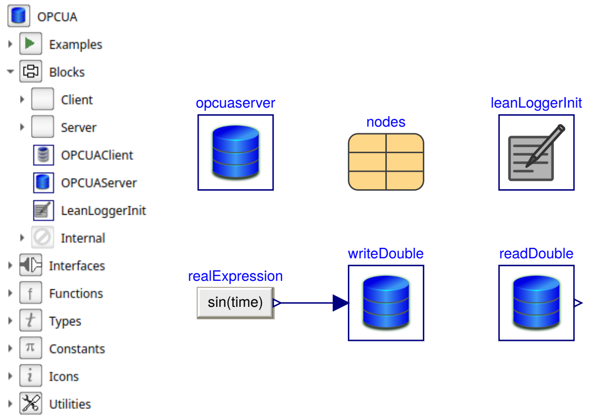


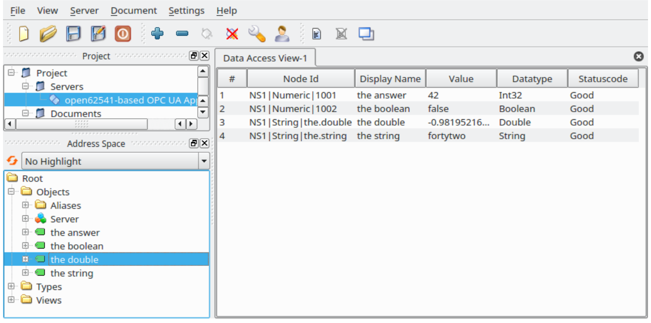
Figure 1. Library structure (left) and basic OPC UA server example (right).

The right side of Figure 1 shows the diagram layer of a basic server example. The `opcuaserver` component is an instance of the `OPCUAServer` block. It is declared as "`inner`", hence it can be accessed in all deeper levels of the model's instance hierarchy as an "`outer`" element. The `nodes` component is a configuration record. It is passed to the `opcuaserver` component as a parameter. It contains information about OPC UA nodes which shall be created on the server during initialization.

At the bottom are instances of blocks for writing and reading variables of type *Double*. These blocks contain parameters which specify the *NodeIds* of the *variable nodes* which are the target (source) of the write (read) operation.

An OPC UA server is started by simply simulating the model. Usually real-time synchronization is desired which can be either provided by the capabilities of the simulation environment or by external code, *e.g.*, using the Modelica_DeviceDrivers library (Thiele et al. 2017). After starting the simulation, the OPC UA server will listen on a specified port for client connections (default port number is 4840).

A good general purpose OPC UA test client is the freely available UaExpert from Unified Automation (*UaExpert* 2021). It can provide a plethora of information in different configurable views. Figure 2 shows a possible view on the connected basic server example. The "Project" pane (upper left window) shows the connected server(s) ("open62451-based OPC UA Application"). The "Address Space" pane (lower left window) allows browsing through the nodes of the server's information model. Nodes from the "Address Space" pane can be drag-and-dropped into the "Data Access View" (DA View) pane (right window). The DA View creates a subscription and



#	Node Id	Display Name	Value	Datatype	Statuscode
1	NS1|Numeric|1001	the answer	42	Int32	Good
2	NS1|Numeric|1002	the boolean	false	Boolean	Good
3	NS1|String|the.double	the double	-0.98195216...	Double	Good
4	NS1|String|the.string	the string	fortytwo	String	Good

Figure 2. Unified Automation's test client UaExpert connected to the basic OPC UA server example.

allows monitoring (configurable) aspects of the nodes. In this example there are four monitored nodes. Only the node with the display name "the double" changes its value during the simulation. This is the node referenced in the `writeDouble` block of Figure 1. The remaining nodes in the DA View are defined in the `nodes` record (including an initial value), but they are not written to during simulation time.

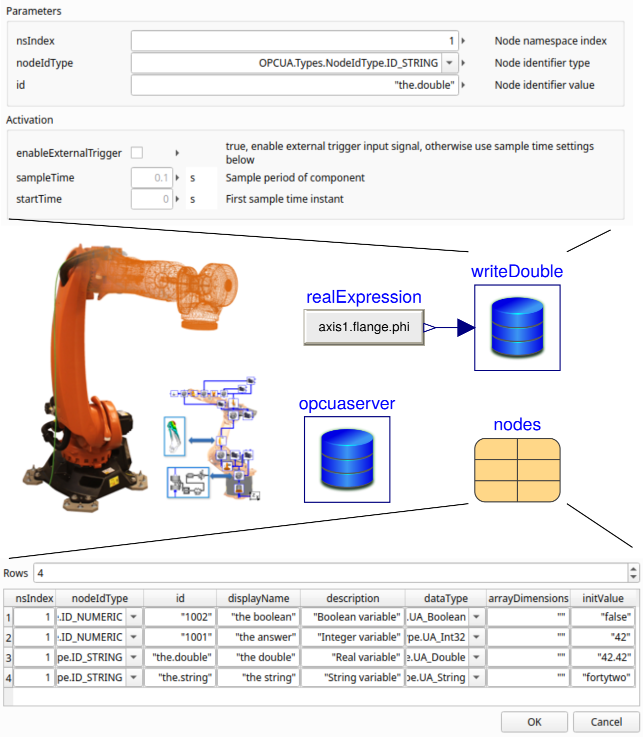
The DA View also shows the pivotal *NodeId* attribute in the "Node Id" column. *NodeIds* refer to a namespace with an additional identifier value that can be an integer, a string, a guid or a bytestring. In the example two nodes are using an integer identifier ("Numeric") and two are using a string identifier ("String"). All shown nodes are in the namespace index "1" ("NS1"), the namespace reserved for the local server.

4 Server and Client

On the one hand the library provides OPC UA server functionality with the goal of modeling the communication interface of (physical) hardware components. The main task is providing simulated process variables to external devices, *e.g.*, for HIL simulation or virtual commissioning. On the other hand the library provides OPC UA client functionality with the goal of querying actual process variables from hardware components, *e.g.*, for process visualization and monitoring or (model-based) high-level control tasks. It is possible to use server and client blocks within the same Modelica model.

4.1 Server

Figure 3 gives more details about some server related blocks. The left-hand side robot denotes a placeholder for an arbitrary physical model with process variables which are published by an OPC UA server running on the physical device. The `nodes` record instance is an approach for collecting *nodes* which shall be created on the server in one central data structure. The `opcuaserver` has two main parameters: `portnumber`, for specifying the server's listening port, and `nodes`, a configuration record for defining own *nodes* and *namespaces* on the server. Hence, the declaration in the model is:


Parameters

nsIndex	1	Node namespace index
nodeIdType	OPCUA.Types.NodeIdType.ID_STRING	Node identifier type
id	"the.double"	Node identifier value

Activation

enableExternalTrigger	□		true, enable external trigger input signal, otherwise use sample time settings below
sampleTime	0.1	s	Sample period of component
startTime	0	s	First sample time instant

Rows 4

	nsIndex	nodeIdType	id	displayName	description	dataType	arrayDimensions	initValue
1	1	:.ID_NUMERIC	"1002"	"the boolean"	"Boolean variable"	.UA_Boolean	""	"false"
2	1	:.ID_NUMERIC	"1001"	"the answer"	"Integer variable"	'pe.UA_Int32	""	"42"
3	1	pe.ID_STRING	"the.double"	"the double"	"Real variable"	:.UA_Double	""	"42.42"
4	1	pe.ID_STRING	"the.string"	"the string"	"String variable"	pe.UA_String	""	"fortytwo"

Figure 3. OPC UA server: The `node` record specifies a list of nodes which are created on the server. Other blocks, like `writeDouble`, can use these *NodeIds*.

```
inner Blocks.OPCUAServer opcuaserver(
    portNumber=4840, nodes=nodes);
```

The `writeDouble` block needs to specify a *NodeId* (using parameters `nsIndex`, `nodeIdType`, `id`) which identifies the node to which it periodically writes its input[1]. It is an error if this node does not exist on the server or if it is not compatible.

The record instance `nodes` is an instance of `OPCUA.Types.Nodes`. Its structure is shown in Listing 1. The annotations are hints to editing tools for creating a convenient graphical user interfaces (GUI) for filling the variable sized arrays, *e.g.*, the dialog for the `VariableNode vars[:]` array is the one displayed at the bottom of Figure 3.

Listing 1. Nodes configuration record.

```
record Nodes
  String nsUris[:] = fill("", 0) annotation
       (Dialog(enable=true));
  VariableNode vars[:] = fill(
      OPCUA.Types.VariableNode(), 0)
      annotation (Dialog(enable=true));
end Nodes;
```

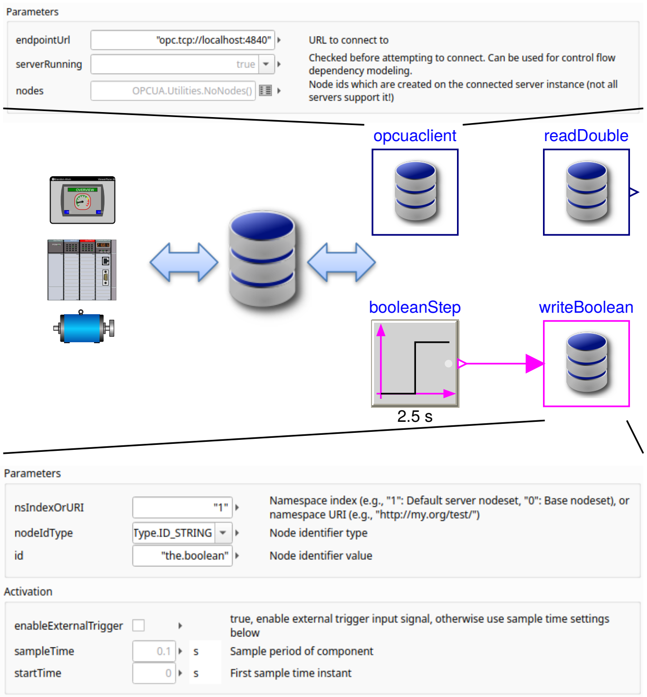
There are limits and compromises in this approach. First most, it cannot be used to create arbitrary OPC UA

[1] Parameter `nsIndex` identifies the namespace index ("1" denoting the namespace reserved for the local server), `nodeIdType` the *NodeId* type (here either "Numeric" or "String"), `id` the identifier value, hence these parameters correspond to the attributes displayed in the "Node Id" column of Figure 2.

nodes. Instead, it aims at supporting a subset of *variable node* types which have a rather straightforward mapping to primitive Modelica types (including arrays of these types). Also notice, that it uses strings at places where this seems not to fit in all cases (`id`, `arrayDimension`, `initValue`). This is a compromise so that the columns can encode values of different data types, *e.g.*, `Boolean`, `Integer`, or `Real`.

4.2 Client

Figure 4 gives more details about some client related blocks. The left-hand side denotes the server side to which the client connects, which typically provides hardware related process variables.



Parameters

endpointUrl	"opc.tcp://localhost:4840"	URL to connect to
serverRunning	true	Checked before attempting to connect. Can be used for control flow dependency modeling.
nodes	OPCUA.Utilities.NoNodes()	Node ids which are created on the connected server instance (not all servers support it!)

Parameters

nsIndexOrURI	"1"	Namespace index (e.g., "1": Default server nodeset, "0": Base nodeset), or namespace URI (e.g., "http://my.org/test/")
nodeIdType	Type.ID_STRING	Node identifier type
id	"the.boolean"	Node identifier value

Activation

enableExternalTrigger	□		true, enable external trigger input signal, otherwise use sample time settings below
sampleTime	0.1	s	Sample period of component
startTime	0	s	First sample time instant

Figure 4. OPC UA Client: The `opcuaclient` block connects to an OPC UA server. Blocks like `writeBoolean` can access nodes on the server by their *NodeId*.

The `opcuaclient` needs to specify the endpoint URL of the server instance. In the example a local server listening at port 4840 is expected. Variable `serverRunning` is not a `parameter`, it has continuous-time variability. Its main use is in Modelica models which combine `opcuaclient` and `opcuaserver` blocks in one model. In this case, it can be used to ensure that the server is ready, before the client (in the same model) connects to it. Given the respective access rights, it is also possible to create new nodes on the server. For this purpose a configuration record can be passed as parameter `nodes` (default: no nodes are created). Identical to the server case the record needs to be an instance of `OPCUA.Types.Nodes`.

Often the main interest is in reading process variables, but it is also possible to write data to the server as indicated by the `writeBoolean` block. Comparing Figure 4 with Figure 3 shows that the client and server blocks for accessing variables have a similar interface.

Sometimes *NodeIds* used by the server are known a priori by the client. This is for example the case for standardized information models, including the OPC UA specification itself, as well as companion specifications, or vendor specific information models. However, in practice the client often has no a priori knowledge of the *NodeIds* used by the server for variables of interest. Instead, the client browses the address space of the server programmatically in order to find *NodeIds* corresponding to objects and variables of interest. This is feasible since the address space is represented hierarchically, allowing for simple and complex structures to be discovered and utilized by OPC clients (see the “Address Space” pane in Figure 2).

In particular, nodes in the address space can be discovered by *browse paths*, *i.e.*, by following a sequence of named references (*browse names*) from a start node to hierarchically subordinated nodes. A basic starting node for searching is the root objects folder. Its *NodeId* is known, because it is defined by OPC UA specification. The presented Modelica library browse paths delimited by ‘/’ can be used for discovering *NodeIds*, *e.g.*, starting from the root objects folder the browse path “Server/ServerStatus/State” can be used for retrieving the *NodeId* assigned to the status code variable of the server.

There are different types of references which can model different types of hierarchical composition, *e.g.*, for modeling component composition, folder organization, or instance hierarchies. This allows a fine-grained filtering based on the type of reference. However, at present the DLR OPCUA library does not discern between different types of references when trying to resolve a browse path.

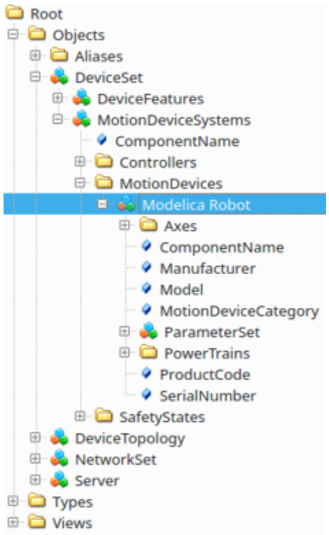
4.3 OPC UA for Robotics

Since a lot of work in our group is centered around robotic applications, integrating the OPC UA for Robotics Companion Specification (OPC 40010-1 2019) is of high interest. The robotics companion specification itself depends on a companion specification featuring an information model for devices (OPC 10000-100 2020). Integrating these companion specifications is a considerable effort and there are different possible approaches.

Accompanying to the textual specification documents there exist XML-based information model definitions according to the OPC UA Nodeset XML schema. These so called *nodeset files* encode OPC UA information models and are understood by respective tools. The open62451 distribution includes an *XML Nodeset Compiler*, a python-based tool, which can generate C code (including C header files) from such XML files. This C code needs to be included in the build process for compiling working server applications. Hence, for supporting the desired companion specifications it is required to modify the build process so that code is generated from the respective nodeset files and this code needs to be included in the compilation process.

The present prototype uses an approach in which a C++ wrapper of the robotics information model encapsulates required function calls to the “low-level” interface of the underlying open62451 library. All code, including dependencies to generated code from the XML Nodeset Compiler and the open62541 library, is assembled in one shared library (see section 5 for more details). The developed C++ classes themselves are encapsulated by a plain external C interface which is compliant to the Modelica external function interface. These C functions are called from respective Modelica functions and are used for creating Modelica external objects.

The Modelica functions can then be used for creating an OPC UA for Robotics compliant information model on the server. Figure 5 shows an example of such an information model as seen by a connected client.



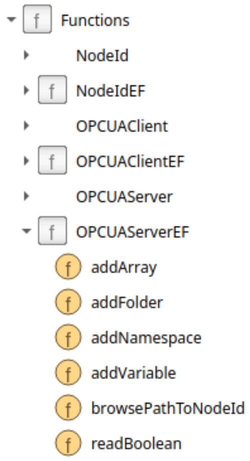
- Root
  - Objects
    - Aliases
    - DeviceSet
      - DeviceFeatures
      - MotionDeviceSystems
        - ComponentName
        - Controllers
        - MotionDevices
          - Modelica Robot
            - Axes
            - ComponentName
            - Manufacturer
            - Model
            - MotionDeviceCategory
            - ParameterSet
            - PowerTrains
            - ProductCode
            - SerialNumber
        - SafetyStates
    - DeviceTopology
    - NetworkSet
    - Server
  - Types
  - Views

Figure 5. OPC UA for Robotics compliant server example as seen by a connected test client (UaExpert).

5 Function Interface

Figure 6 shows an excerpt from the comprehensive `Functions` package. This package contains definitions for external objects and functions operating on these objects.

Table 1 lists several notable external objects and their underlying (open62451) data structures. The open62451 data structures can be recognized by the library’s naming convention of using the prefix “`UA_`” for its exported symbols. Modelica external objects are opaque pointers to some address in memory, so (in principle) the underlying C data structures can be changed or extended without the need of changing the Modelica code. Possible changes may even include to swap out the underlying OPC UA library (though there is at present no intention for such a step).



- Functions
  - NodeId
  - NodeIdEF
  - OPCUAClient
  - OPCUAClientEF
  - OPCUAServer
  - OPCUAServerEF
    - addArray
    - addFolder
    - addNamespace
    - addVariable
    - browsePathToNodeId
    - readBoolean

Figure 6. Excerpt of the `Functions` package. It contains external object classes and external functions (EF) operating on these objects.

Table 1. Notable external objects and their opaque pointer mappings.

Modelica	*C/C++ → open62451 (UA_...)*
`NodeId`	→ `UA_NodeId`
`OPCUAClient`	→ `UA_Client`

`OPCUAServer`	→ C++ struct with server related settings:
	`struct uam_server {`
	`  std::thread threadID;`
	`  UA_Server *server;`
	`  UA_Boolean running;`
	`  uam_PubSub *pubSub;`
	`};`
	Particularly, it includes a member of type pointer to `UA_Server`.

While `NodeId` and `OPCUAClient` are directly mapped to open62541 data structures, the `OPCUAServer` external object is mapped to a C++ wrapper structure which contains additional information. After a configuration phase, the server loop is started in a dedicated thread by a function called `OPCUAServer.run(...)`. The identifier of the spawned thread is saved in the struct member `threadID` and struct member `running` is set to “true”.

The struct member `pubSub` is a composite object which aggregates data structures and logic related to the OPC UA Publish/Subscribe (PubSub) extension. PubSub extends the OPC UA client/server architecture with facilities which (among other things) can enable low-latency communication. Results of an experimental low-latency open62451 PubSub implementation are reported in (Pfrommer et al. 2018). The PubSub extension is usable as an experimental feature within the DLR OPC UA library, but it is not yet intended for real-world use-cases.

A view on the DLR OPC UA library’s layered architecture is shown in Figure 7. Only core components are shown, components like the logging facilities or experimental components are suppressed.

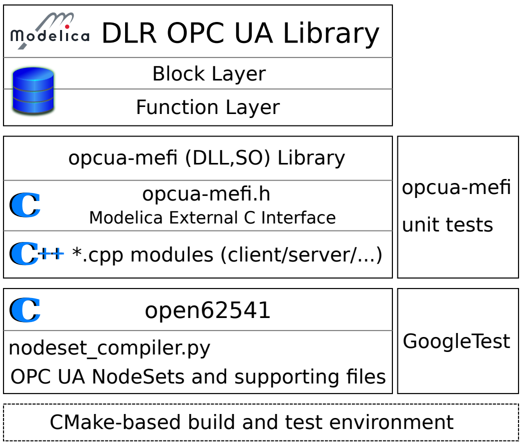

Modelica
DLR OPC UA Library
Block Layer
Function Layer
opcua-mefi (DLL,SO) Library
C
opcua-mefi.h
Modelica External C Interface
C++
*.cpp modules (client/server/...)
opcua-mefi
unit tests
C
open62541
nodeset_compiler.py
OPC UA NodeSets and supporting files
GoogleTest
CMake-based build and test environment

Figure 7. DLR OPC UA library's layered architecture.

The project build, test and packaging automation is managed by the CMake cross-platform family of tools (*CMake* 2021).

The top Modelica OPC UA library uses the Modelica external C functions interface (MEFI) of the underlying *opcua-mefi* library. The *opcua-mefi* library is a dynamic link library (DLL) on Windows, or a shared object library (SO) on Linux. Its Modelica external function interface compliant application programming interface (API) is declared in the header file *opcua-mefi.h*.

While the *opcua-mefi* library declares a Modelica compliant external C function interface, the internal library consists of C++ code. This code wraps and adapts open62451 facilities into structures which can be conveniently used from Modelica. One may say it provides a Modelica-oriented high-level interface to a subset of the open62451 library. Although invisible to a Modelica library user, it is actually the most extensive part of the DLR OPC UA library.

An important part of the *opcua-mefi* library is its unit tests. These tests use the GoogleTest framework (*GoogleTest* 2021) and its CMake integration in order to provide a convenient testing environment on the supported platforms. It integrates nicely with various development environments, *e.g.*, JetBrain's CLion or Microsoft Visual Studio.

The base technology stack is provided by the open62451 open-source library from the open62451 project (*open62541* 2021). It is an impressive open-source C (C99) implementation of OPC UA, licensed under the liberal Mozilla Public License v2.0. Despite its good documentation and a large set of indispensable examples, there is a considerable learning curve for using the library. Though a good part of the learning curve can be attributed to the inherently large and complex OPC UA standard itself.

6 Application Example

The Factory of the Future project (DLR 2021) is a cross-sectoral research project within the German Aerospace Center (DLR). The aim is to develop a wide range of digital production technologies, robotic systems and robotic applications for flexible and networked manufacturing processes, and to demonstrate them in 'lead scenarios'.

One cross-sectoral scenario which is investigated is an assembly process for a motor saw. The scenario includes (physical) robot cells from the Institute of Robotics and Mechatronics (DLR-RM) and the Center for Lightweight-Production-Technology (DLR-ZLP). OPC UA is used as interoperability standard between the different robot cells and involved institutes. The task of our institute, the Institute of System Dynamics and Control (DLR-SR), is the modeling of the assembly process with appropriate fidelity. Our goal with this work is to explore digital twin applications based on physically accurate models.

Modelica is used as modeling language for the physics-based digital twin. There are several challenges for enabling the intended applications, among them:

- Modeling the assembly process requires efficient object manipulation capabilities which can accommodate real-time data updates.
- The Modelica-based simulation model needs to connect and synchronize with the real-world entities and processes.

The first issue lead to the development of a new Modelica library for manipulation tasks, which is outside the scope of this work (Reiser 2021). The present work is concerned with finding a solution to the second issue.

As a first step towards the complete assembly process, one robot cell has been connected at the time of this writing. Figure 8 shows the considered robot cell which has been set up in DLR-RM's lab. The depicted robot is from the recent generation of DLR-RM's light-weight robots and bears the project name SARA (Safe Autonomous

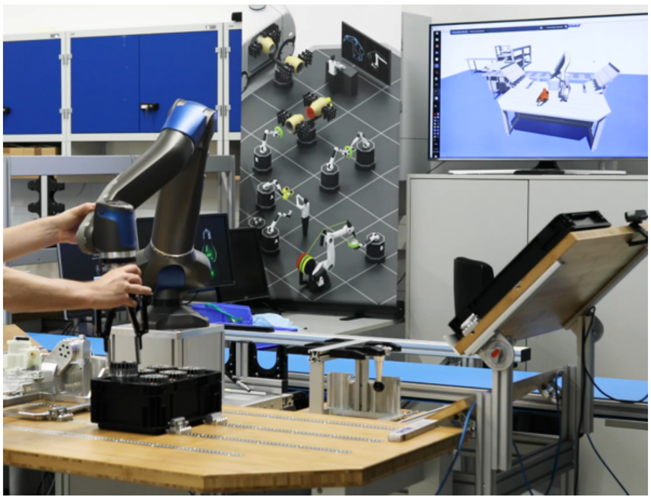
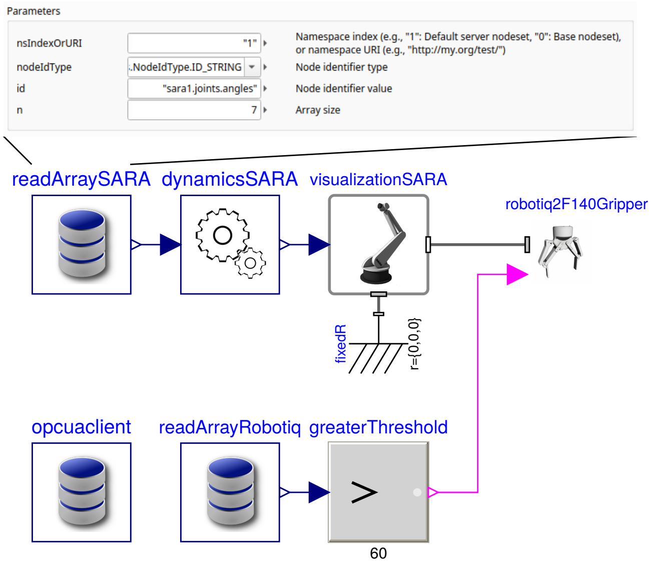


Figure 8. Robot cell from DLR-RM synchronized with Modelica-based real-time simulation model (upper-right screen) from DLR-SR using the DLR OPC UA library for connectivity.

Robotic Assistant), (Iskandar et al. 2020). In the upper-right corner a screen shows a visualization of DLR-SR's simulation model (*i.e.*, the "digital twin") which is synchronized with real-time data from the SARA robot cell.

Figure 9 shows the OPC UA related excerpt of the Modelica model used for the demonstration depicted in Figure 8. There are two OPC UA client blocks which read server variables stemming from the SARA robot cell: `readArraySARA` reads the joint angles of the robot arm, `readArrayRobotiq` reads the position of the gripper. At present, a simple threshold is used for giving an indication whether the gripper is closed or open. Besides the OPC UA blocks, the figure also shows a composite block with a (simple) dynamics model for the SARA robot, as well as a block which is responsible for the 3D visualization of the robot arm.



Parameters

nsIndexOrURI	"1"	Namespace index (e.g., "1": Default server nodeset, "0": Base nodeset), or namespace URI (e.g., "http://my.org/test/")
nodeIdType	.NodeIdType.ID_STRING	Node identifier type
id	"sara1.joints.angles"	Node identifier value
n	7	Array size

Figure 9. Excerpt of the OPC UA related parts of the Modelica model used for the demonstration depicted in Figure 8.

A rather complex block shown in this excerpt of the complete model is the gripper block. It implements the manipulation mechanics and interacts with the assembly parts. The assembly parts, as well as the conveyors and tables in the scene are not shown. All those parts are not in the scope of this work, they are part of the aforementioned library for manipulation tasks. In addition, some visualization related blocks are suppressed. The visualization is provided by a prototype of the next generation of the DLR Visualization library (Hellerer, Bellmann, and Schlegel 2014; Kümper, Hellerer, and Bellmann 2021).

Block `readArrayRobotiq` reads an array variable, but the array has only one element (the gripper position, a value in the dimensionless range $[0,100]$). Essential parameters of the `readArraySARA` block are shown at the top. Notice, that a simple OPC UA approach is used in which the seven joint angles are packed into one array which is identified by a statically fixed string-based *NodeId*. Therefore, neither the OPC UA Robotics extension described in subsection 4.3 is used, nor is there any need for sophisticated node discovery mechanics on the client side.

In summary, the described demonstration was a step towards the envisioned Factory of the Future scenario. In particular, it showed the feasibility of using OPC UA as interoperability standard. Since the DLR OPC UA library also supports OPC UA server functionality (see subsection 4.1), it was possible during development to model the SARA robot cell including its OPC UA server interface and connect it with the client application of Figure 9 within the same model. This simplifies application development, because there is no need that the actual robotic hardware is available. Further work, extending the presented base functionality for exploring more complete digital twin related scenarios is ongoing.

7 Discussion

At the beginning of this library development effort there was the long-term vision of (automagically) generating Modelica models from devices described in established or future automation standards. Since OPC UA is an important standard for the communication aspect, the idea was to explore the generation of Modelica models from OPC UA information models, encoded in *nodeset files* (*i.e.*, *NodeSet2.xml* files), from automation devices and machinery.

After short initial research into available third party library and tools the idea emerged of developing a simple Modelica-oriented C interface on top of the open62451 API. This interface should particularly support the fundamental data types from Modelica (`Boolean`, `Integer`, `Real`, `String` scalars and arrays) and translate between these Modelica types and OPC UA types.

7.1 First Steps

The first steps with the open62451 library were very smooth thanks to good documentation (including working examples) and a polished CMake-based build system. However, striving for more general OPC UA support, including some more advanced constructions, quickly becomes more intricate.

OPC UA defines low-level aspects, like *Int16*, *UInt32*, *Int64*, which cannot always be mapped satisfactorily to Modelica (*e.g.*, the Modelica external function interface defines that `Integer` are mapped to C `int`, hence signed 32-bit integers on common Linux and Windows platforms). For not being overly restrictive on the allowed OPC UA variable types (potentially unsafe) conversions are used at respective places in the *opcua-mefi* library. For mitigation, safe variable value ranges can be checked dynamically in the C code and runtime errors can be risen when violations are detected.

Besides OPC UA built-in types which have a rather straightforward mapping[2], where also exist built-in types

[2] Modelica `Boolean`: OPC UA *Boolean*; Modelica `Integer`:

with no such mapping, e.g., *NodeId* to Modelica. These types require additional design decisions, *e.g.*, *NodeId* is mapped to a Modelica external object. Other OPC UA built-in types, *e.g.*, *XmlElement*, are simply not supported at this state.

7.2 Refactoring

Furthermore, there is a huge flexibility how respective variable nodes can be defined or discovered in OPC UA and often very similar (but not identical) functions and patterns are used for achieving a certain task on either the server or the client. This lead to quite a lot of repetitive code in the *opcua-mefi* library which at some point was addressed by using C++ templates and its code generation facilities for achieving more generic and succinct code.

On top of this first interface an experimental Modelica code generator was written which takes a *NodeSet2.xml* file as input and generates a skeleton of Modelica code with the intent of simplifying and accelerating the development of component simulations and physics-based digital twins. While this worked for the very limited number of elements considered for the experiment, it also became apparent that a more complete (industry-relevant) Modelica code generator could hardly be based on the facilities of the present *opcua-mefi* library.

7.3 Another Approach Needed?

The open62541 library itself uses code generation at various places for providing an API which can encompass the comprehensive OPC UA standard. The key here is that OPC UA standard information is not only English text, but partly already encoded in machine processable files, most notable, *NodeSet2.xml* files. One could compare these to a "standard library" in programming which itself is based on more fundamental principles (syntax and semantics of the underlying programming language). Hence, using an appropriate mechanization, C code can be generated from relevant machine processable files.

This could also be key for enabling a more generic and complete Modelica interface. Instead of the high-level oriented API of the *opcua-mefi* library, one could try to interface the lower-level open62451 more directly and use code generation techniques for gaining a Modelica function interface which is closer to the open62451 API.

Although it seems unrealistic to expect that this would magically solve all problems, a clever approach in this direction could push the limits.

7.4 Domain-Specific Extensions

Instead of striving for a level of generality which would allow taking a *NodeSet2.xml* file and generate suitable Modelica code, another option is to manually develop library support for selected (standardized) domain-specific information models of interest. Although it might be a significant initial development effort for supporting a new domain, it can result in well-thought-out reusable library blocks for quickly modeling devices which adhere to the standardized domain-specific interface.

This is the approach used for the integration of the Robotics Companion Specification as described in subsection 4.3. Compared to the generic approach, it is easier to achieve and can be a good alternative if the expected use-cases adhere to such domain-specific information models.

7.5 Outside of Modelica

Another approach with a different angle is using dedicated automation-oriented simulation platforms and rather import Modelica models. Using the Functional Mock-up Interface (FMI) standard for such a purpose suggests itself. For example, Hensel et al. (2016) explore an approach of integrating FMI-based co-simulation with the SIMIT simulation platform from Siemens using OPC UA as a generic middleware technology.

Using a dedicated integration platform can be a practical and flexible alternative if a such a platform is available. The discussed Modelica library approach might be more appealing in Modelica-centric development processes, or if using an additional platform seems too costly or complicated[3].

8 Conclusions and Outlook

Work on the presented Modelica library was started with no prior experience with OPC UA technology. Thanks to available resources, like the open-source open62451 project, or the freely available OPC UA test client from Unified Automation GmbH, the first steps were rather smooth and quick.

However, moving to slightly more advanced concepts it quickly became apparent that the OPC UA standard and related tooling has an intimidating complexity, and it took longer towards the current state of the Modelica library with a more moderate progress than expected.

Indeed, there are plenty of more OPC UA features and aspects which are not yet explored or implemented within the library, or simply not covered for not exceeding the scope of this paper. Among them, supporting the PubSub extension, which has been briefly mentioned in section 5. OPC UA PubSub extends the applicability of OPC UA beyond a strict client/server model and also sketches a direction towards low-latency communication schemes (Pfrommer et al. 2018). These are hot topics with no final conclusion and ongoing discussions within standardization bodies (Bruckner et al. 2019).

A good amount of the motivation for this work is based on the anticipation that OPC UA will play a crucial role for future automation systems. In this respect, exploration of the underlying concepts and technology has been an im-

OPC UA *SByte*, *Byte*, *Int16*, *UInt16*, *Int32*, *UInt32*, *Int64*, *UInt64*; Modelica `Double`: OPC UA *Float*, *Double*; Modelica `String`: OPC UA *String*.

[3] Notice that it is still possible to export a Modelica model with OPC UA interface blocks as Functional Mock-up Unit (FMU) and import it into a co-simulation environment.

portant driving factor in the development. Consequently, the library has the status of an experimental in-house technology prototype. So far, its runtime stability has been pleasantly reliable (credits to the open62451 project), but the interface, structure, naming, documentation, and the supported feature set is not fixed, yet. Nevertheless, the library can be made available to interested partners.

Future plans with the library include further exploration of more advanced concepts, as well as following (and potentially integrating results of) ongoing standardization efforts with a particular interest for robotic applications and real-time industrial communication.

Acknowledgements

This work was partially funded by the DLR project Factory of the Future. The author would like to thank Robert Reiser and Tobias Bellmann for feedback and support during the development of the library. For the described application example the author would like to thank all who contributed to the demonstration. Particularly, Robert Reiser who implemented the Modelica application example model, Oliver Eiberger and Timo Bachmann who provided CAD data of the robot cell, and Korbinian Nottensteiner and Stefan Schneyer who implemented the OPC UA interface of the robot cell. Finally, the author would like to thank the reviewers for the constructive comments to improve the manuscript.

References

Bruckner, Dietmar, Marius-Petru Stănică, Richard Blair, Sebastian Schriegel, Stephan Kehrer, Maik Seewald, and Thilo Sauter (2019). “An Introduction to OPC UA TSN for Industrial Communication Systems”. In: *Proceedings of the IEEE* 107.6, pp. 1121–1131. DOI: 10.1109/JPROC.2018.2888703.

CMake (2021). URL: https://cmake.org/ (visited on 2021-04-15).

DLR (2021). *DLR Factory of the Future*. URL: https://factory-of-the-future.dlr.de/ (visited on 2021-04-15).

ESI Group (2021). *SimulationX OPC-UA Client Modelica library*. URL: https://doc.simulationx.com/4.2/1033/Content/Libraries / InterfacesGeneral / Communication / OPCUA / open62541.htm (visited on 2021-06-24).

Fritzson, Peter et al. (2020). “The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development”. In: *Modeling, Identification and Control* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

GoogleTest (2021). URL: https://github.com/google/googletest/ (visited on 2021-04-15).

Harrison, William S. and Frederick Proctor (2015). “Virtual Fusion: State of the Art in Component Simulation/Emulation for Manufacturing”. In: *Procedia Manufacturing* 1. 43rd North American Manufacturing Research Conference, NAMRC 43, 8-12 June 2015, UNC Charlotte, North Carolina, United States, pp. 110–121. ISSN: 2351-9789. DOI: 10.1016/j.promfg.2015.09.069.

Hellerer, Matthias, Tobias Bellmann, and Florian Schlegel (2014). “The DLR Visualization Library - Recent development and applications”. In: 10th *Int. Modelica Conference*. Ed. by Hubertus Tummescheit and Karl-Erik Årzén. Lund, Sweden. DOI: 10.3384/ECP14096899.

Hensel, Stephan, Markus Graube, Leon Urbas, Till Heinzerling, and Mathias Oppelt (2016). “Co-simulation with OPC UA”. In: *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*. Poitiers, France, pp. 20–25. DOI: 10.1109/INDIN.2016.7819127.

Iskandar, Maged, Christian Ott, Oliver Eiberger, Manuel Keppler, Alin Albu-Schäffer, and Alexander Dietrich (2020). “Joint-Level Control of the DLR Lightweight Robot SARA”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8903–8910. URL: https://elib.dlr.de/138637/.

Kümper, Sebastian, Matthias Hellerer, and Tobias Bellmann (2021). “DLR Visualization 2 Library - Real-Time Graphical Environments for Virtual Commissioning”. In: 14th *Int. Modelica Conference*. Ed. by Martin Sjölund, Adrian Pop, Lena Buffoni, and Lennart Ochel.

OPC 10000-100 (2020). *OPC Unified Architecture – Part 100: Devices*. Tech. rep. Release 1.02.02. OPC Foundation.

OPC 40010-1 (2019). *OPC UA for Robotics Companion Specification Part 1: Vertical integration*. Tech. rep. OPC Foundation.

open62541 (2021). URL: http://open62541.org (visited on 2021-04-15).

Pauker, Florian, Thomas Frühwirth, Burkhard Kittl, and Wolfgang Kastner (2016). “A Systematic Approach to OPC UA Information Model Design”. In: *Procedia CIRP* 57. Factories of the Future in the digital environment - Proceedings of the 49th CIRP Conference on Manufacturing Systems, pp. 321–326. ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.2016.11.056.

Pfrommer, Julius, Andreas Ebner, Siddharth Ravikumar, and Bhagath Karunakaran (2018). “Open Source OPC UA PubSub Over TSN for Realtime Industrial Communication”. In: *23rd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. Turin, Italy, pp. 1087–1090. DOI: 10.1109/ETFA.2018.8502479.

Profanter, Stefan, Ari Breitkreuz, Markus Rickert, and Alois Knoll (2019). “A Hardware-Agnostic OPC UA Skill Model for Robot Manipulators and Tools”. In: *24th IEEE International Conference on Emerging Technologies And Factory Automation (ETFA)*. IEEE. Zaragoza, Spain. DOI: 10.1109/ETFA.2019.8869205.

Reiser, Robert (2021). “Object Manipulation and Assembly in Modelica”. In: 14th *Int. Modelica Conference*. Ed. by Martin Sjölund, Adrian Pop, Lena Buffoni, and Lennart Ochel.

Sjölund, Martin and Adeel Asghar (2018). *Real-time debugging and monitoring*. Technical Note D4.2 (M36). ITEA3, Project 14018: OPENCPS project.

Thiele, Bernhard, Thomas Beutlich, Volker Waurich, Martin Sjölund, and Tobias Bellmann (2017). “Towards a Standard-Conform, Platform-Generic and Feature-Rich Modelica Device Drivers Library”. In: 12th *Int. Modelica Conference*. Ed. by Jiří Kofránek and Francesco Casella. Prague, Czech Republic. DOI: 10.3384/ecp17132713.

UaExpert (2021). URL: https://www.unified-automation.com/products/development-tools/uaexpert.html (visited on 2021-04-15).

Wolfram Research (2021). *Wolfram OPCUA Modelica library*. URL: https://reference.wolfram.com/system-modeler/libraries/OPCUA/OPCUA.html (visited on 2021-06-24).

ZVEI (2015). *The Reference Architectural Model Industrie 4.0 (RAMI 4.0)*. Tech. rep. Zentralverband Elektrotechnik- und Elektronikindustrie e.V. (ZVEI).

A Modelica library for Thermal-Runaway Propagation in Lithium-Ion Batteries

Christian Groß Andrey W. Golubkov

Virtual Vehicle Research GmbH, Austria, batterysafety@v2c2.at

Abstract

Based on the Thermal Runaway (TR) experiments conducted in our laboratory a simple method of describing a battery's thermal behaviour was developed. In the approach - which we call *simple tracing method* - the temperature rate measurement from Accelerating Rate Calorimetry (ARC) during a TR experiment is approximated to determine the thermal behaviour of the model. This method was implemented in Modelica using Dymola. Alongside the implementation of the TR model a complete Modelica package with useful models for TR propagation simulation was developed. The package called **"BatterySafety"** serves as a foundation for further model development to investigate TR propagation and physical counter measures in greater detail. The focus of all the models in the package was efficiency, intelligibility and user-friendliness. With this approach we are able to simulate TR propagation of a complete battery pack.

Keywords: Lithium-Ion Batteries, Battery Safety, Thermal Runaway, Thermal Runaway Propagation

1 Introduction

List of Acronyms

- TR - Thermal Runaway
- LIB - Lithium-Ion Battery
- ARC - Accelerating Rate Calorimetry
- SOC - State-Of-Charge

Motivation

Batteries of electric vehicles are safe at temperatures below ≈ 80°C. If Li-ion cells reach temperatures above 80°C the cells start to degenerate and when they reach a even higher temperatures they may transit into thermal runaway (TR). An over temperature of a cell is a major safety concern and may be caused by failures inside the cell or by the surrounding of the cell. In automotive applications, where energy storage usually does not consist of a single cell but a battery pack of multiple connected cells, TR is an even bigger concern. The battery pack allows for TR propagation, where failure in one cell causes failure in adjacent cells, which can ignite the whole battery pack. Understanding and predicting TR and TR propagation are key to the development of countermeasures.

Feng, Lu, et al. (2016) said, what could not have been said better: "Experimental study on TR propagation is essential, and we may need massive experiments on TR propagation to help design a safe battery pack. [...] However, given that the experimental study on TR propagation within a battery pack costs much time and money, building an easy-to-use, verified abuse model that realistically captures the mechanisms of TR propagation in battery pack is beneficial to find efficient approaches to prevent TR propagation." In that spirit the *BatterySafety* library was developed, backed by the knowledge our team gathered over the years (Andrey W. Golubkov et al. 2018; Essl, Andrey W. Golubkov, et al. 2020) and will be released under the Modelica License 2.

Paper Structure

This paper consists of six sections, starting with the introduction in section 1. In section 2 we are going to look at the mathematical backbone of the library and introduce the simple tracing model. Section 3 then discusses the implementation of the library with a focus on key models. And section 4 showcases the functionality of the library by looking at some example simulations, followed by section 5 and section 6 the discussion and conclusion of the paper.

2 Simple Tracing Model

The key element of the *BatterySafety* library is the TR model around which the library is built. This section aims to educate about our modelling approach, it's benefits and weaknesses. First we lay out the foundation, then we process that into a implementable form and last we talk about how the heating rate data needed for the model is obtained.

2.1 Deriving the model

Consider the thermal runaway being a simple chemical reaction

$$A \to B \tag{1}$$

with the concentration c of the educt A going from 1 to 0

$$c = 1 \to c = 0 \tag{2}$$

The chemical reaction happens inside a Li-ion cell with a fixed heat capacity C_p and an overall enthalpy change ΔH. The temperature rate of the Li-ion cell during TR depends on the kinetics of the chemical reaction $\dot{c}$ and on the heat exchange with the ambient $\alpha_A \Delta T$ (α_A is the heat transfer coefficient times the area).

$$\dot{T} = -\frac{\Delta H}{C_p}\,\dot{c}(T) + \alpha_A \Delta T \tag{3}$$

which is a rough approximation of the first law of thermodynamics. Consider a ideal adiabatic reaction with $\alpha = 0$; no heat exchange with the ambient

$$\dot{T} = -\frac{\Delta H}{C_p}\,\dot{c}(T) \tag{4}$$

Rewrite and consider the temperature dependence of $\dot{T}$

$$\dot{c}(T) = -\frac{C_p}{\Delta H}\,\dot{T}(T) \tag{5}$$

The temperature rate $\dot{T}(T)$ is given by the piecewise linearisation in the rate plot of the experiment, as depicted in Figure 1. In the rate plot $g(T)$ follows the measured temperature between 150°C and 300°C. The measured decrease of the rate at temperatures $T > 300$°C is ignored in $g(T)$, because it is caused by spatial effects (spatial heat flow during the experiment). At $T < 150$°C the self heating could not be detected by the measurement equipment (because it is obscured by the external heating at > 1°C/min), but it is known from other publications (Feng, Zheng, et al. 2019) that the slope can extrapolated below 150°C.

During simulation the model traces $\dot{T}(T)$: it starts with some elevated temperature, extracts the $\dot{T}(T)$ from the rate plot and updates the temperature. In the next time step the model extracts $\dot{T}(T)$ with the updated temperature and so on until all educts are consumed ($c = 0$) then the rate is forced to zero $\dot{T}(c \leq 0) = 0$.

$$\dot{T}(T) = \begin{cases} \text{linear approximation}, & \text{if } c > 0 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

The overall enthalpy change ΔH is calculated from the maximal temperature change during the experiment

$$\Delta H = C_p\,(T_{max} - T_{onset}) \tag{7}$$

To successfully and efficiently implement the simple tracing model in Modelica, we need to adapt the equations a little. Instead of considering the concentration c, we want to think in terms of available energy E, which we assume to depend linearly on the c

$$E(T) = \Delta H c(T) \tag{8}$$

We can now express Equation 5 in terms of energy as well

$$\dot{E}(T) = -C_p \dot{T}(T) \tag{9}$$

which can be implemented in a straight forward way as Section 3.2 shows.

The simple tracing model has two main disadvantages

- The heat loss to the ambient is not considered, therefore the values of $\dot{T}(T)$ and ΔH may be underestimated. The actual error depends on the thermal insulation in experiment setup, e.g. the deviation from ideal adiabatic condition.
- The the piecewise linear heating rate $\dot{T}(T)$ gives no additional insight into the chemical reactions system (order of the reaction, number of Arrhenius reactions)

The big advantage of this approach is that no curve fitting is needed.

2.2 Thermal-Runaway Experiment

The data we use to generate the temperature rate approximation is gathered first-hand from conducting TR experiments. To prepare the experiment we first attach temperature sensors on the cell's surface and then apply insulating cover. The cell assembly then is put into the sample holder, which is fitted with heating elements. Through springs the sample holder applies a desired force on the cell. Further the sample holder is inserted into the reactor, the reactor sealed pressure tight and flushed with N_2 gas. Last the cell is cycled (dis- and then recharged) and charged to desired state-of-charge (SOC). We then conduct the experiment and let the cell transit into TR by chosen method. Usually TR is induced by heating the cell up to the cell-dependent TR onset temperature. The interested reader can find an in-depth explanation on our method and setup in our other publications. We recommend the work of Essl, A. Golubkov, and Fuchs (2020) and Essl, Andrey W. Golubkov, et al. (2020) as a starting point. For the examples in section 4 we have used data from "Cell #2" in the paper of Essl, A. Golubkov, and Fuchs (2020).

3 Library Implementation

The *BatterySafety* library features too many models to explain them all in detail in this paper. Therefore we decided to just show the implementation of the core models from the bottom up. We are now going to look at how we implemented the TR model and arrived at the cell, module and battery pack level models.

3.1 Energy Storage

At the end of section 2 the equations were expressed in terms of energy, because we wanted to captivate this idea in Modelica. The absolute nature of the stored energy proved to be a surprisingly challenging obstacle in implementation. We started by implementing a light green coloured energy connector

Listing 1. Definition of the energy connector

```
connector EnergyPort
    Modelica.SIunits.Energy
        E "Energy at Port";
    flow Modelica.SIunits.EnergyFlowRate P
        "Energy flow rate (power)";
```

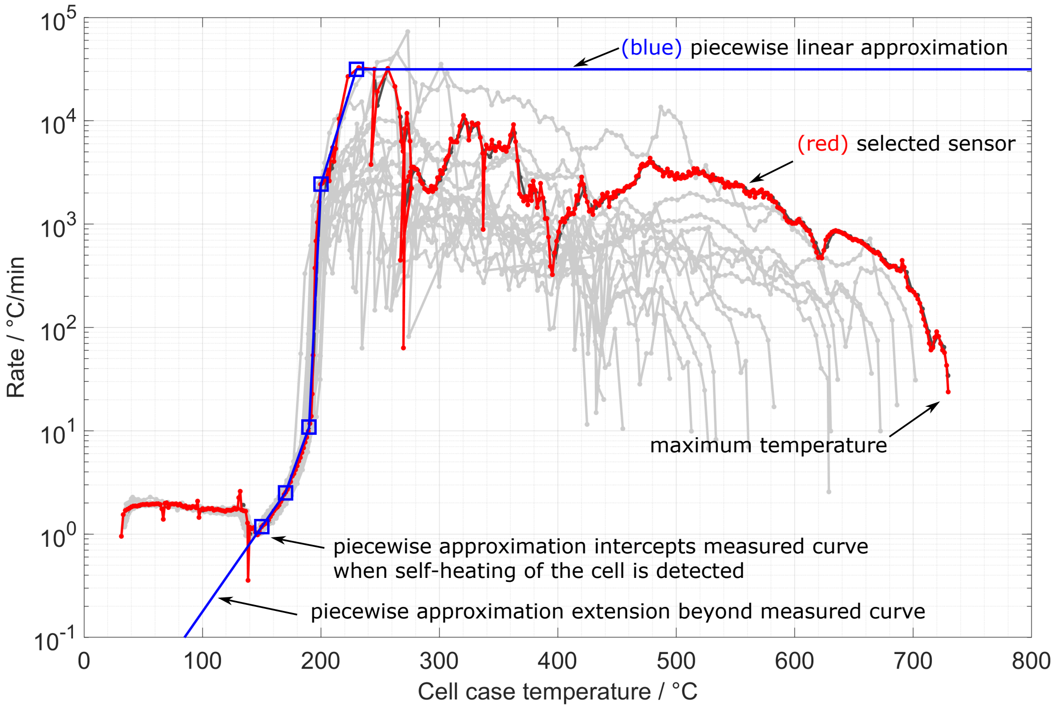


Figure 1. Temperature rate measurements from a single experiment: (red) Selected measurement for modelling, (blue) the linear approximation in logarithmic scale and (grey) measurements from different points on the cell's surface.

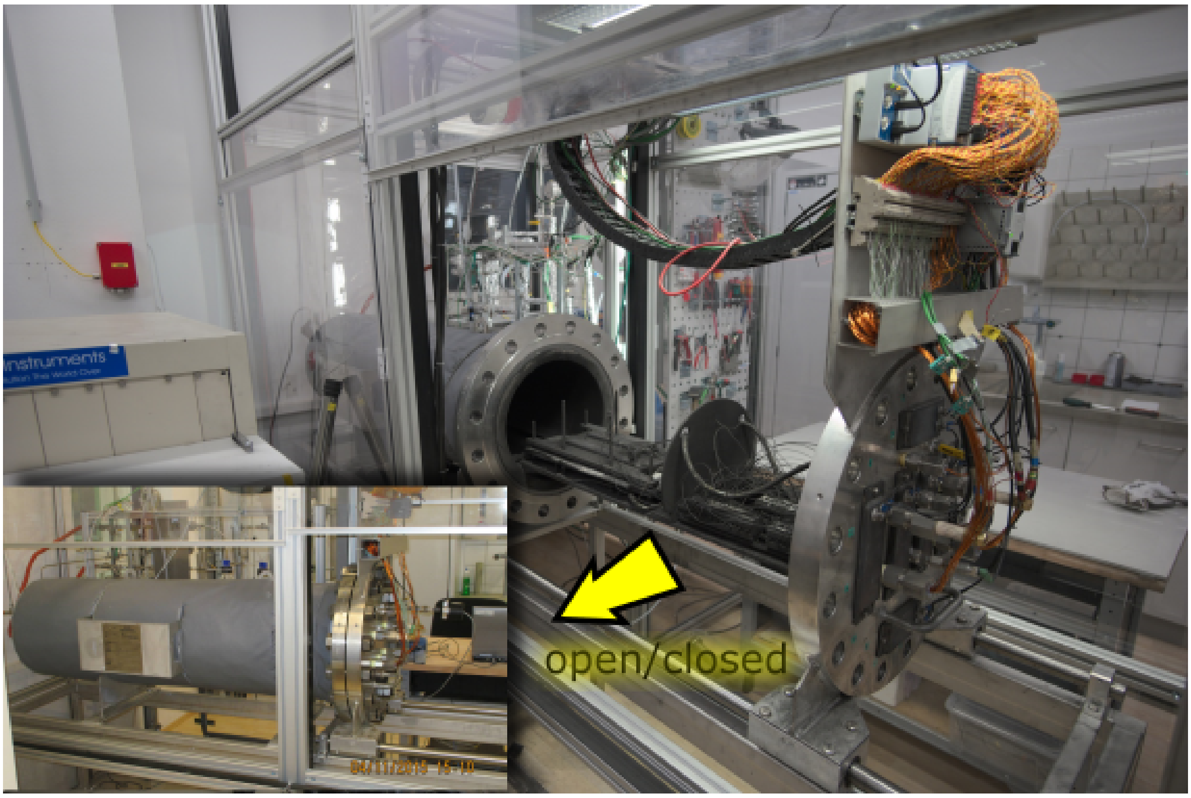


Figure 2. Sealable reactor for thermal runaway experiments

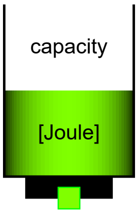


Figure 3. Icon of the EnergyStorage model

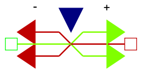


Figure 4. Icon of the VariableEnergy2HeatConversion model

and an energy storage model analogous to a capacitor with a fixed flow rate constant of 1.

Listing 2. Equation section of the EnergyStorage model

```
equation:
    E = port.E;
    der(E) = port.P;
```

All attempts at inhibiting energy outflow when the energy reached 0, within the model, were unsuccessful. Therefore this was handled in the model converting the energy from the storage to heat, at the rate given by the real input *u*.

Listing 3. Equation section of the VariableEnergy2HeatConversion model

```
equation:
    h.port.Q_flow =
        if e_port.E<=0 and u>0 then 0
        else -u;
```

3.2 Thermal Runaway Model

With the models for storing and converting energy it is easy to translate Equation 9 into the *ChemicalHeatGeneration* Modelica model as Figure 5 shows. The *temperatureSensor* feeds the *thermal_port* temperature into the *HeatEmissionFunction*, which corresponds to the *linear approximation* in Equation 6. Then *power_a* takes the result to the 10th power, resulting in the conversion rate for *conversion2heat* and thus draining the *energyStorage* and releasing heat. The library also features a second heat release model, extending the one explained here. In the extended model a boolean output was added to indicate when the output of the *HeatEmissionFunction* is greater than a certain threshold. This was done to have a reliable way of telling other models in the system, that the cell has transited into TR.

3.3 Cell Model

Figure 6 shows the cell model, to ease implementation it is based on a prismatic cell. From top to bottom it's 3 sub-models are:

1. The electric cell model
2. The heat release or thermal-runaway model
3. The heat distribution model

Each of these are replaceable to allow the user to select different model options as they see fit. However, as our focus lay on TR propagation, the included options are limited. There are two electric models in the library, consisting of a voltage source and a resistor. In one the resistor does not generate heat and in the other the resistor generates joule heat. The heat distribution models consist of heat capacitors and heat resistances modelling the cells thermal mass and allowing for either 1D (shown in Figure 10) or pseudo 2D (shown in Figure 8) heat exchange with other models. Furthermore the heat resistances have been adapted to switch to a lower resistance once the cell transits into TR. This was done to accelerate TR propagation, as the venting of hot gas, which plays a major role in TR propagation, has not been implemented yet (Srinivasan et al. 2020). The TR models have been discussed in the previous section. The records in the cell model serve the purpose of providing parameters for the models. These will be discussed later on. The internal heat-port labelled *CellTemperature* allows to manually read out the cell temperature more conveniently. This also allows to have an external model heat or cool the cell from the inside, if desired. At this point we want to mention that some simple models for heating and cooling are implemented in the library. The heaters are for the most part modelled after their real life counterpart we use in our laboratory, but in a simplistic way. These are used to let the cell transit into TR. The methods of cooling were implemented as a proof of concept but without having a foundation in reality. This was done to gain the experience necessary to implement a thorough model in the future. For the purpose of this paper we will not explain these models in more depth, the interested reader is advised to look at the mentioned models in the package.

3.4 Module and Pack Models

Once the cell model is completed and allows for thermal connection in multiple directions, modelling battery-modules (Figure 7) and packs becomes a matter of drawing connections. Figure 8 and Figure 9 show how a module is made from connected cells and how a battery pack is made from several connected modules.

3.5 Parameter Records

As mentioned in subsection 3.3, we use records to provide parameters for the models in the package. This provides two advantages. First it reduces the chances of parameterisation errors, as a parameter set for any model has to be entered just once. This also makes correcting such errors easier as there is only one spot where to look. Second, parameters can be changed with a few clicks for many instances of a model at the same time. How powerful these advantages are is evident when looking at the module and pack models in subsection 3.4. In the battery pack are 30 instances of the same module, in each module 12 instances of the same cell and each cell needs 20 parameters. That

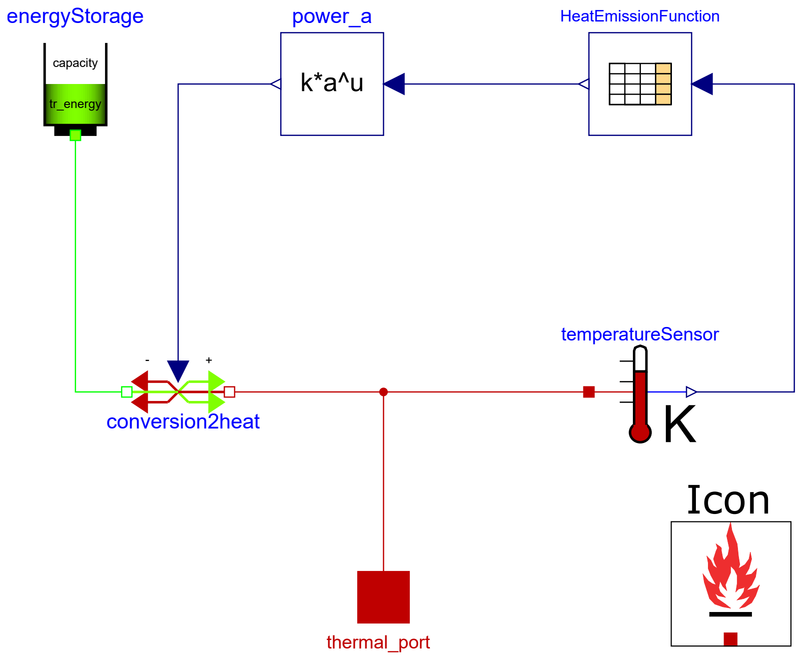


Figure 5. Diagram view and Icon of the *ChemicalHeatGeneration* model

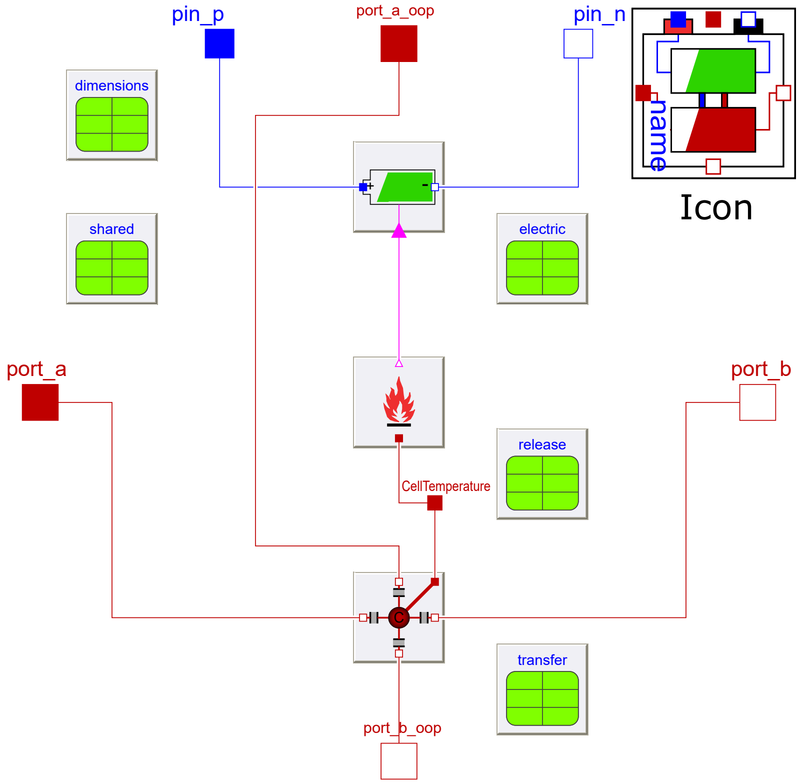


Figure 6. Diagram view and Icon of the cell model. The thermal connectors port_a and port_b transfer heat parallel to the electrode stack (in plane), while port_a_oop and port_b_oop transfer heat orthogonal to the stack (out-of-plane)

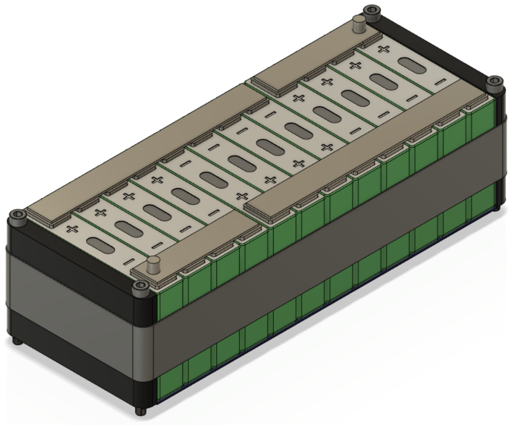


Figure 7. Sketch of a module consisting of 12 Li-ion cells with prismatic casing stacked in a row and electrically connected in 4p3s configuration (4 parallel, 3 series).

means manually setting the parameters of all the cells requires 7200 inputs.

To further aid the user, the records are defined in groups fitted to the cell sub-model they intend to provide parameters for. Parameters shared between two or more submodels are grouped in a separate record definition. All the records needed to fully parametrise a cell are embedded in another record definition. Each record definition has some data-filled records stored within the library.

4 Examples

Now we want to demonstrate the capability of the library with three examples, from single cell to battery pack level. For each example the simulation setup is described and the simulation result is shown. How and to which extend these results can be interpreted will be addressed in section 5.

4.1 Single Cell

For our first example we want to show, that our model correctly captures the thermal behaviour of a LIB. Therefore this example features a single cell with one sided heating as Figure 10 shows. The second heater is turned off and is just there so that the released heat has somewhere to go, which makes the temperature curve look a bit more natural. Figure 11 shows the result, which when compared to measurement data indicates that the dynamics of TR have been captured correctly by our model.

4.2 Module

With the second example we want to highlight the scalability of our model. In essence the setup remains the same, but instead of a single cell a module of 12 cells is heated from one side (Figure 12). This setup models the module in a insulated, robust and undamaged casing (no heat exchange with the casing) in an atmosphere devoid of oxygen where no air may enter. The resulting cell temperatures, as shown in Figure 13, display TR propagation through the entire module. This reassures that the developed TR model is capable of portraying this kind of behaviour.

4.3 Pack

The last example in this paper concerns the simulation of an entire battery pack, or traction battery, capable of powering an electric car. This was done to show that the model is applicable on this scale with reasonable computation time. Figure 9 shows the layout of the pack, consisting of an array of 3 × 10 modules and TR is initiated by heating the cell in the lower left corner. Each module contains 12 cells for a total of 360 cells. Same as for the module example we consider the pack to be encased and the atmosphere devoid of oxygen. The amount of cells simulated make it hard to grasp the full picture just by line-plotting temperature versus time. Therefore we used a *MATLAB* script to turn the simulation data into a video. This video depicts all of the cells positions and temperatures by colour throughout the entire duration of the simulation. Figure 14 shows some of the videos frames depicting the propagation of the TR throughout the pack.

5 Discussion

To summarise: The *BatterySafety* library features a TR model for LIB based on the *simple-tracing model* discussed in section 2. This TR model is then coupled with a simple electrical model and a heat conduction based heat transport model to form a cell model. Cell models are connected to form a module model and module models are connected to form a model of a battery pack. Heater models are used to heat an initial cell until transiting into TR. The cooling models included in the library are at an early stage of development and should only be seen as a proof of concept. The combination of the *simple-tracing model* with Modelica allows for fast simulations of TR propagation on the scale of a battery pack.

However additional work needs to be done to ensure the validity of the results. At the current state we only have data from cells in a constant volume N_2 atmosphere with the same initial pressure. This data is valid to use for simulations considering an intact case of a module or battery pack, that can withstand TR at constant volume. We believe this to be a realistic scenario in the case of a non-crash related thermal event happening in a battery pack. We have found no literature how in N_2 atmosphere different initial pressures affect thermal behaviour, either at constant volume or constant pressure. There is literature on the effects of different initial pressures in constant pressure air environment affecting heat released during TR, with higher heat release at higher pressure(Xie et al. 2020; Chen et al. 2019). However we believe this can not be extrapolated to the case of N_2 atmosphere, as we see the cause of the higher heat release at higher pressure in the higher availability of oxygen. The effects of different initial pressures in N_2 atmosphere on the heat released during TR in constant pressure and constant volume environments is a topic for research in the near future.

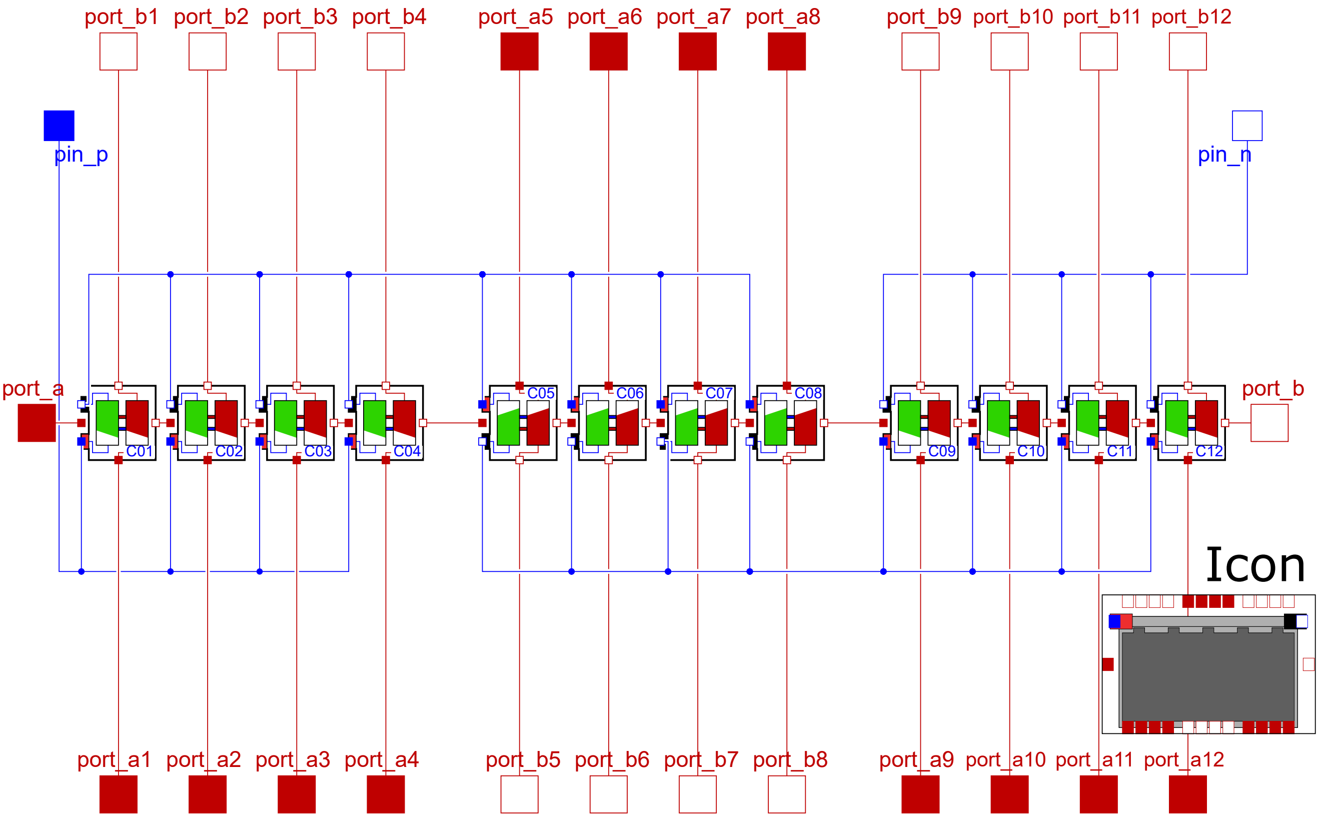


Figure 8. Diagram view of a 12 cell module. The 12 cells (C01 ... C12) are geometrically stacked in a row. Each 4 cells are electrically connected in parallel and the 3 electric cluster are electrically connected in series (4p3s configuration).

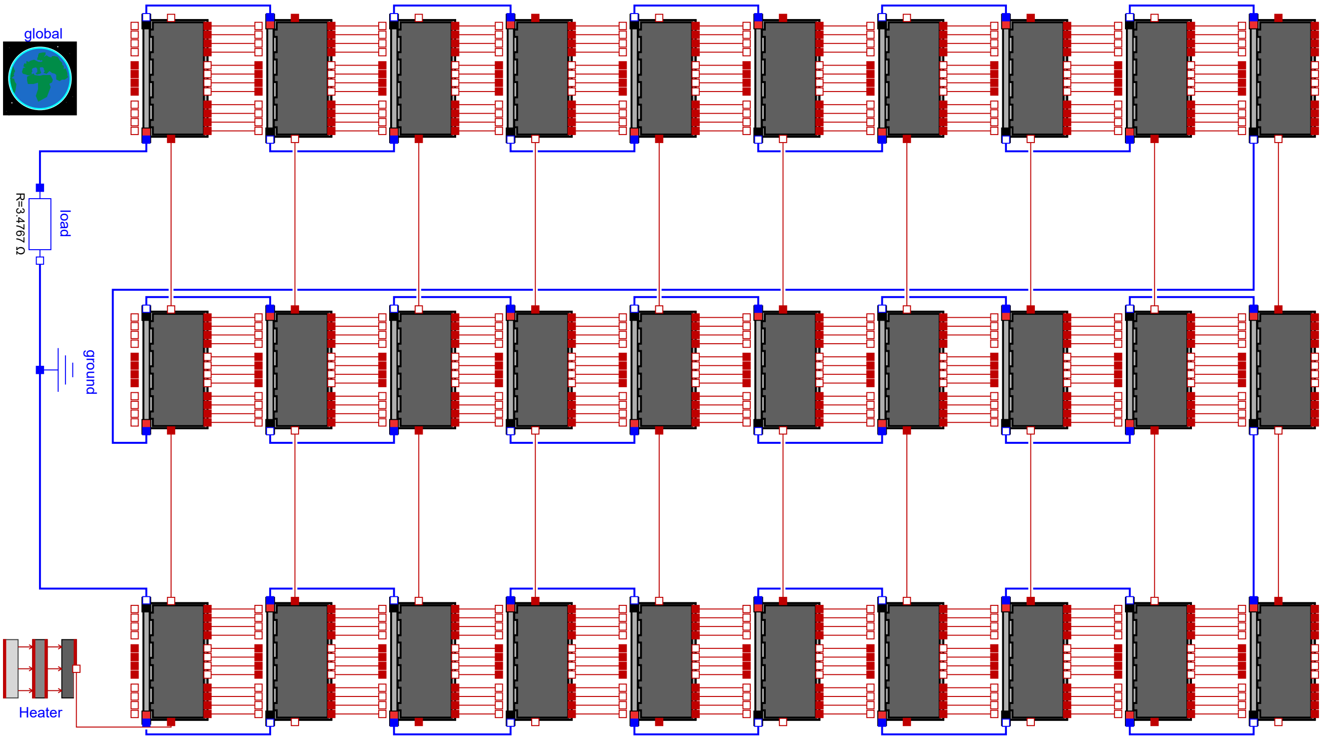


Figure 9. Diagram view of a battery pack. The pack consists of 30 modules which are geometrically arranged in a 3 × 10 pattern. All modules are electrically connected in series. The electric loop is closed by a resistor which represents the main electric motor in an electric car.

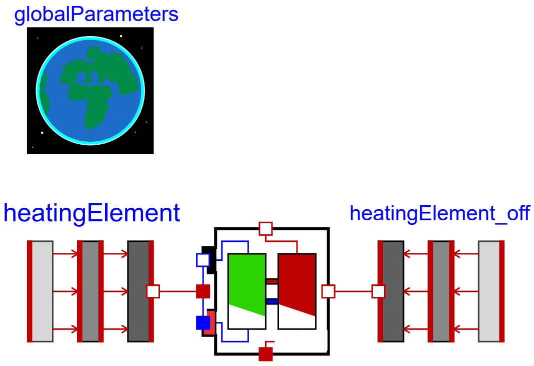


Figure 10. Diagram view of the single cell example. The cell is sandwiched between two heater plates.

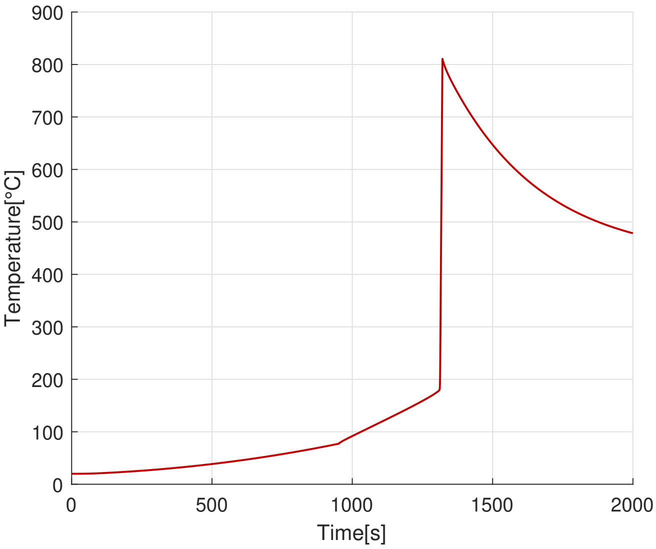


Figure 11. Temperature of a simulated cell. The cell is heated by one of the heater plates until - at 1300 s - the cell transits into TR. During the TR the cell reaches a temperature above 700°C. After the TR the cell starts to cool down slowly.

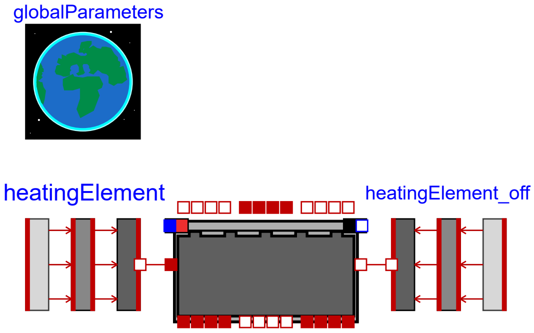


Figure 12. Diagram view of module example. Here the module is sandwiched between two heater plates.

We are aware that a major factor contributing to propagation speed is likely to be the release of high velocity hot gas (venting) as a cell transits into TR (Srinivasan et al. 2020), which is not considered in the library yet. To counteract this fact the cell models feature variable heat resistors, which lower their thermal resistance once the cell transits into TR. This is easy to implement and computationally less expensive compared to adding models for gas release. Whether this approach leads to reliable and generalizable simulation results, or if the implementation of gas release is absolutely necessary has to be determined. Therefore a comparison of these two approaches, as as well as verifying the model will be the topic of a paper in the near future. Further topics worth investigating may be:

- Implementing and researching the effects of active cooling measures and other thermal safety features.
- Implementing interactions between a cells electric model and the energy storage for SOC dependent heat release and more accurate electric behaviour.
- Short-circuits and electric arching between modules, which can be caused by conductive particles released by cells during TR.
- Combustion of vent gases with air, when air enters the battery pack.
- Coupling the model with a FEM cell level simulation to more accurately capture heat distribution in the cell where TR first is initiated.
- Integrating the model in a complete vehicle co-simulation
- Finding a way to estimate the temperature rate curve without the data from a TR experiment
- Finding a relationship between known cell parameters and TR onset temperature.

6 Conclusion

We introduced the simple-tracing model for TR in LIB and implemented it in a Modelica package called *BatterySafety* alongside several other models, to study and further our understanding in TR propagation. Through the discussed examples we have shown the applicability of the developed package to realistic use-cases. The set of such cases is small for now but will continue to grow as research progresses and the package is updated.

The package is easy to use and the models computationally inexpensive even on the scale of a battery pack found in electric vehicles. This allows even inexperienced users to use our TR model to simulate a range of TR propagation simulations with little effort and time and will be available to everyone under the Modelica License 2. To our knowledge the capability to simulate TR propagation through

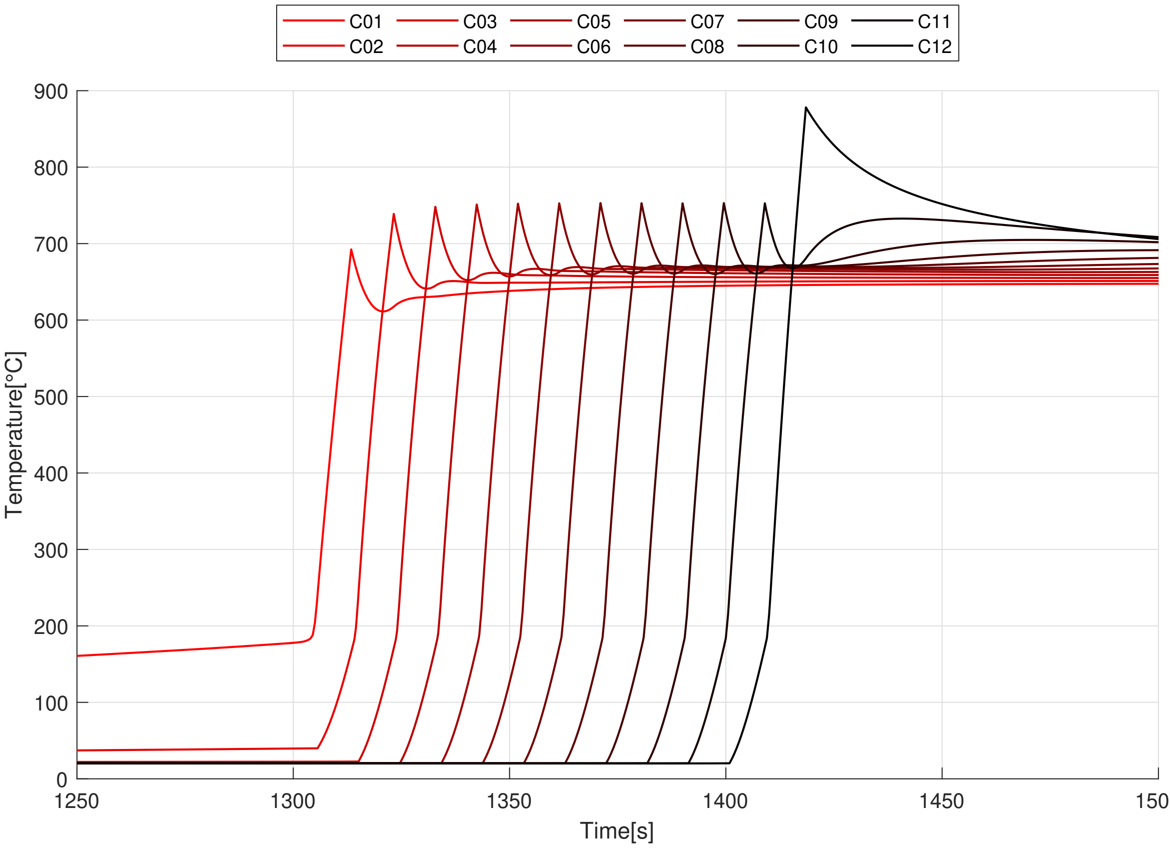


Figure 13. Temperature curves of a simulated module during TR propagation. Each curve belongs to one cell. The TR propagation starts with cell C01 and finishes with the cell C12.

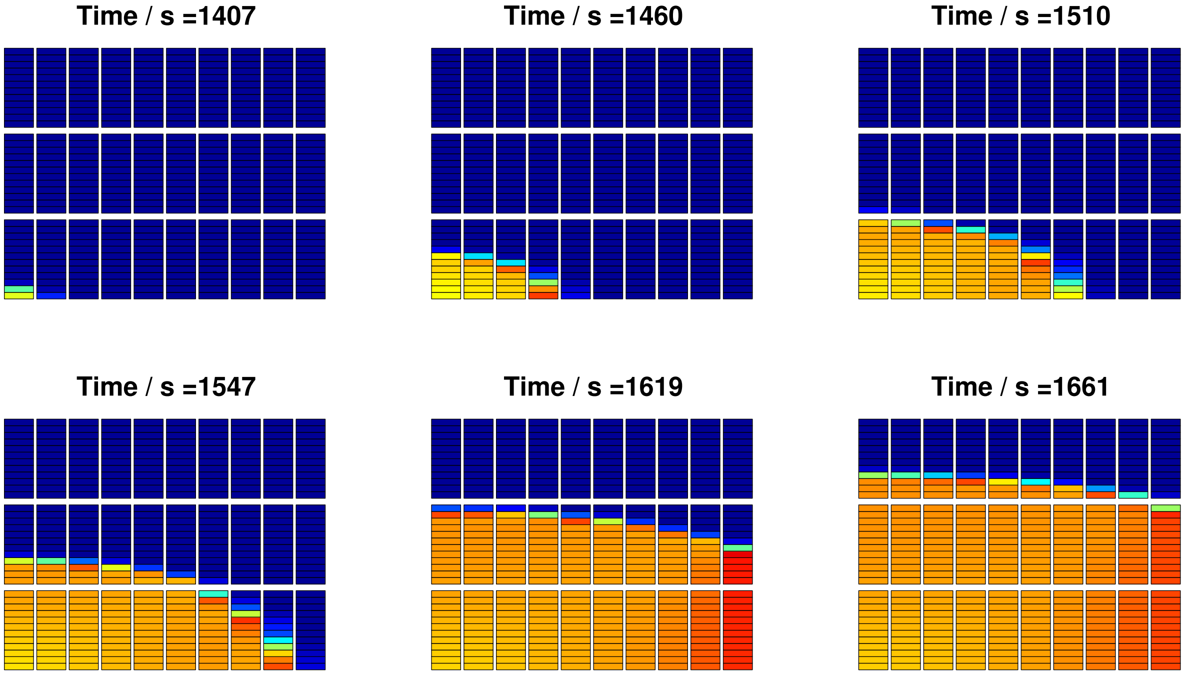


Figure 14. Visualising the TR propagation in a simulated battery pack. Each small rectangle represents a cell. Each stack of cells represents a module. The TR starts in the bottom left cell. With time the TR propagates from cell to cell inside the modules and from module to module.

a battery pack in a reasonable time is novel. Complex 3D simulations with FEM could take weeks or months for such a simulation in contrast to the 1-4 hours our model takes. Looking back at the requirements on the model from Feng, Lu, et al. (2016) in section 1 we can conclude that with the work done so far, we are about halfway to realising viable TR propagation simulations on a battery pack level.

We consider this paper and the *BatterySafety* library as the foundation to enable fast, low-skill, virtual prototyping for LIB in the future. We hope that the scientific community will take notice of the value Modelica offers to LIB research and aid us in improving upon our work. We believe this package can help to design safer battery packs in the future.

Acknowledgements

This publication was written at Virtual Vehicle Research GmbH in Graz. The authors would like to acknowledge the financial support within the COMET K2-Competence Center for Excellent Technologies Program of the Federal Ministry for Transport, Innovation and Technology (BMVIT), the Federal Ministry for Digital and Economic Affairs (BMDW), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG).

Conflicts of interest

The authors declare no conflicts of interest.

References

Chen, Mingyi et al. (2019). "Environmental pressure effects on thermal runaway and fire behaviors of lithium-ion battery with different cathodes and state of charge". In: *Process Safety and Environmental Protection* 130, pp. 250–256. ISSN: 0957-5820. DOI: https://doi.org/10.1016/j.psep.2019.08.023.

Essl, Christiane, Andrey W. Golubkov, et al. (2020). "Comprehensive Hazard Analysis of Failing Automotive Lithium-Ion Batteries in Overtemperature Experiments". In: *Batteries* 6.2. ISSN: 2313-0105. DOI: 10.3390/batteries6020030.

Essl, Christiane, AW Golubkov, and Anton Fuchs (2020). "Comparing Different Thermal Runaway Triggers for Two Automotive Lithium-Ion Battery Cell Types". In: *Journal of the Electrochemical Society* 167.13, p. 130542. DOI: 10.1149/1945-7111/abbe5a.

Feng, Xuning, Languang Lu, et al. (2016). "A 3D thermal runaway propagation model for a large format lithium ion battery module". In: *Energy* 115, pp. 194–208. ISSN: 0360-5442. DOI: 10.1016/j.energy.2016.08.094.

Feng, Xuning, Siqi Zheng, et al. (2019). "Investigating the thermal runaway mechanisms of lithium-ion batteries based on thermal analysis database". In: *Applied Energy* 246, pp. 53–64. ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2019.04.009.

Golubkov, Andrey W. et al. (2018). "Thermal runaway of large automotive Li-ion batteries". In: *RSC Advances* 8.70, pp. 40172–40186. ISSN: 20462069. DOI: 10 . 1039 / C8RA06458J.

Srinivasan, Rengaswamy et al. (2020-02). "Preventing Cell-to-Cell Propagation of Thermal Runaway in Lithium-Ion Batteries". In: *Journal of The Electrochemical Society* 167.2, p. 020559. DOI: 10.1149/1945-7111/ab6ff0.

Xie, Song et al. (2020). "Influence of cycling aging and ambient pressure on the thermal safety features of lithium-ion battery". In: *Journal of Power Sources* 448, p. 227425. ISSN: 0378-7753. DOI: https://doi.org/10.1016/j.jpowsour.2019.227425.

The DLR ThermoFluidStream Library

Dirk Zimmer[1] Michael Meißner[1] Niels Weber[1]

[1]German Aerospace Center (DLR), Germany
{dirk.zimmer,michael.meissner,niels.weber}@dlr.de

Abstract

This paper introduces the DLR Thermofluid Stream Library: a free open-source library for the robust modeling of complex thermofluid architectures. Designed to be easy to use, easy to adapt and enriched by a number of examples, this library contains the fundamental components for many different applications such as thermal management of electric cars, power plants, or building physics.

Keywords: Thermal fluids, Thermal process systems, Robust modeling, Heat exchangers, Open source software

1 Introduction

Streams of thermofluids form the basis of many natural and technical systems. Power plants, environmental control systems, refrigerators can all be represented as thermodynamic processes where components such as pumps, heat exchangers, or valves manipulate the stream of a working fluid flowing from the component inlet to its outlet. A computationally very attractive formulation is to express this coupling of inlets and outlets by using algebraic equations in the form:

$$\Theta_{\text{out}} = g(\Theta_{\text{in}}, \dot{m}, \mathbf{x}) \tag{1}$$

with Θ being a tuple representing the thermodynamic state of the medium, $\dot{m}$ being the mass-flow rate and $\mathbf{x}$ the internal state vector of the component (e.g. rotational speed of a pump).

Such an algebraic coupling (as simple as it may be) implies a dramatic idealization of the actual underlying physical system. For instance, we implicitly assume that upstream changes in pressure and enthalpy take immediate effect downstream. Not only may such an assumption be completely inadequate under certain circumstances also these idealizations give rise to highly non-linear equation systems where there might be multiple solutions or none at all. Even if there is a unique solution, it may be very difficult to retrieve.

For these reasons, the modeling of thermofluid streams is rarely performed in purely algebraic fashion. The modeler may want to break the algebraic system down to feasible complexity (for instance by adding volume elements), the modeler may want to model thermal time-constants (for instance by adding volume elements) or transport delay (for instance by adding volume elements). However, for many applications, there is something more clever than just adding volume elements. The library we present in this paper implements such an alternative that is favorable for many applications. It enables the modeler to formulate a system of thermofluid streams in a robustly solvable form using only a few additional state variables.

While robustness is one major design target for the library, the ability to adapt the library to specific needs is another one. The set of potential applications is very large and its elements may differ in their underlying assumptions. Hence any concise set of components cannot realistically be expected to provide full coverage. Instead, the library shall provide solid base components that are easy to read and that can then be adapted to the specific needs that are raised by the user's application field.

Robustness and adaptiveness are hence the two main points what shall make the library competitive with already existing solution in this domain such as other free, commercial, or proprietary Modelica libraries (Casella 2005; Casella et al. 2006; Franke, Casella, Sielemann, et al. 2009; El Hefni and Bouskela 2014), or non-Modelica M&S tools (EcosimPro 2021; Process Systems Enterprise 2021) or even our own previous solutions (Sielemann et al. 2011). These libraries mostly represent either an algebraic modeling style or an ODE modeling style. The paper (Zimmer 2020) offers a comparison of our DAE-based approach to such modeling styles.

But before we recapitulate the underlying robust computational scheme and take a look at the library, let us look at an introductory example.

1.1 Introductory example

Figure 1 contains an example application of an automotive battery, drive-chain and cabin thermal control system. This represents a fairly complex system with several loops and three different media. Amongst other items, there are several switches to change the flow topology and a vapor cycle with two parallel evaporators. Although the example itself is not part of the library, it is entirely build out of its components, some of which internally combine multiple valve models to realize switches for bypasses and loops. For the sake of clarity, we have removed the control elements from the diagram.

The model diagram depicts some important distinctions from the fluid library as part of the Modelica Standard Library (MSL) (Franke, Casella, Sielemann, et al. 2009). First of all, this library features dedicated models for junctions and splitters and abstains from (ab-)using the Modelica connector for this purpose. Second, the de-

DOI
10.3384/ecp21181225

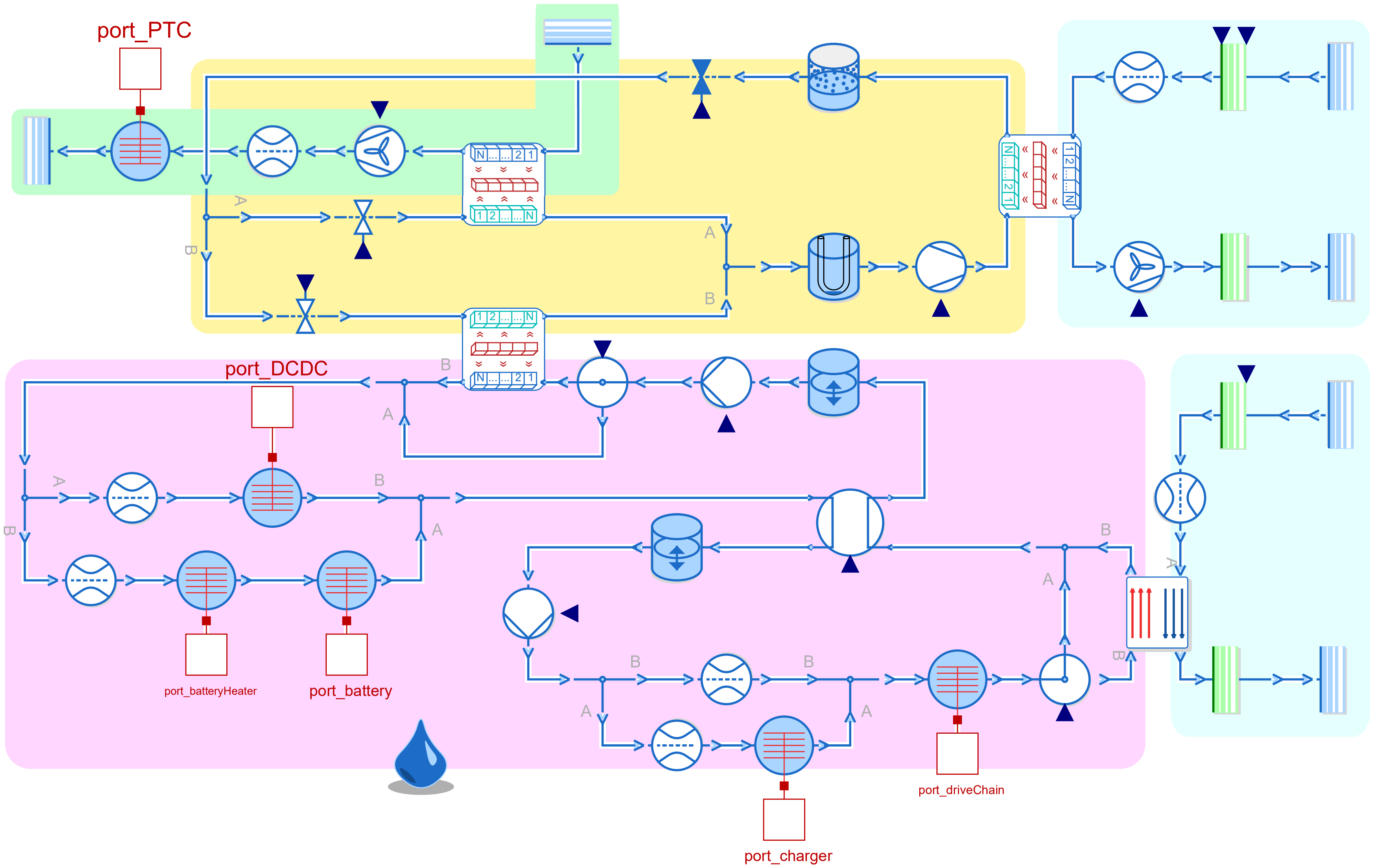


Figure 1. Example simulation of an automotive battery, drive-chain and cabin thermal control system. The glycol-water loop (magenta) is cooling battery, DC-DC converter, charger and drive-chain. A four-way switch can split it into two separate loops instead of the combined loop that is shown. If needed, the loop is cooled by a radiator against fresh air (blue). Additionally it can be cooled with a R134a vapor-cycle (orange). The vapor-cycle is also cooling air going to the vehicle cabin (green) and therefore contains two parallel evaporators, that can independently be shut off by two valves when not in use. It is cooled against fresh air with a condenser. Furthermore, it contains receiver, accumulator and an expansion valve. Battery and cabin-air can be heated with the battery heater and PTC respectively. Both heat exchangeers of the glycol loop can be bypassed if not in use. One of the fresh-air streams and the cabin-air stream is supported by a fan, while the other fresh-air stream is driven purely by dynamic pressure from the vehicle speed.

fault connector type is directional, clearly indicating the direction of the stream. The library offers a solution for undirected flows too but it is supposed to be used only when inevitable. Third, the modeler shall break loops (not merely bypasses, but actual loops) using volume elements. Mostly, this will happen anyway since actual loops need actual reservoirs but cases like the combined loops of liquid cooling with its two reservoirs require an attentive modeler.

The translated model contains of 49 states (eighteen of which are attributed to the discretized heat exchangers of the vapour cycle) and contains only four non-linear systems of size one that can each be locally attributed to one of the four volume components. The initialization problem consists of two linear systems that are manipulated to size zero and poses no problem. The simulation is robust and will run through various topology switches and changing heat-loads, unless one of the media models is driven out of its temperature or pressure ranges.

While some parameterization of all components will have to be made to match measured data of an actual system, most components come with usable default parameters, therefore even complex topologies can be quickly built up and simulate successfully.

2 Robustness

2.1 Underlying methodology and assumptions

As suggested by its name a thermofluid stream is forming the central entity of this modeling approach. The stream is thereby bound either by dedicated boundary models such as sources or sinks or by elements that represent a volume of the medium.

Between these boundaries, the stream consists in a sequence of components such as compressors, valves, etc. that may manipulate the thermodynamic state Θ. Shared among all these components is a common mass-flow rate. So all components alongside a stream uphold the mass-flow balance $\dot{m}_{\text{in}} = -\dot{m}_{\text{out}}$.

The mass-flow rate is always a state variable of the system. Each component of a stream expresses a differential equation of the form:

$$\frac{d\dot{m}}{dt} L = -\Delta r \tag{2}$$

Whereas r is denoted as inertial pressure and L is the inertance of the fluid. Please note that the inertance is solely defined by the geometry of the flow and independent of the thermodynamic state:

$$L = \int ds/A \tag{3}$$

with s being the length of the flow and A its cross-section area. We can make use of the inertial pressure r to decompose the general pressure gradient Δp into

$$\Delta p = \Delta \hat{p} + \Delta r \tag{4}$$

whereas $\hat{p}$ is introduced as steady-mass flow pressure (since $\hat{p} = p$ if $d\dot{m}/dt = 0$). Alongside a stream we approximate the thermodynamic state by using $\hat{p}$ instead of p, accepting a (mostly) small error for unsteady flow conditions. At each boundary we compensate the accumulated difference between $\hat{p}$ and p with the inertial pressure r and accelerate the corresponding mass flow with respect to r.

This leads to a very favorable structure of the equation system where all non-linear computations can be brought into explicit form and the only system of equations in implicit form is strictly linear. Hence a robust solution of the system model can be reliably achieved supposing robust component and media models. More details on the approach and also the handling of junctions and splitters in (Zimmer 2020).

Another way of looking at this approach is that we use different spatial resolutions for $\hat{p}$ and r. Whereas $\hat{p}$ may be resolved for each component, r is only resolved between boundaries of the stream. This is mostly fine because the impact of r on the thermodynamic state is typically low (or even zero for steady flow conditions). However, should the impact of r become vital (unsteady cavitation might be such a case), the modeler is advised to increase the spatial resolution by adding more volume elements at the place of concern.

2.2 Implementation in Modelica

To implement this decomposition, we use a connector that contains a pair of potential and flow variable. The flow is naturally the mass flow rate and the corresponding potential is the inertial pressure since it is the potential variable that determines the dynamics of the flow. The thermodynamic state is then transferred as a signal. Since we are using the standard Modelica.Media library (Casella et al. 2006), using the state record of the media models seems a natural choice.

Listing 1. connectors for directed thermofluid streams

```
connector Inlet
  replaceable package Medium;
  SI.Pressure r;
  flow  SI.MassFlowRate m_flow;
  input Medium.ThermodynamicState state;
end Inlet;

connector Outlet
  replaceable package Medium;
  SI.Pressure r;
  flow  SI.MassFlowRate m_flow;
  output Medium.ThermodynamicState state;
end Outlet;
```

The approach chosen here is thus similar to (Otter et al. 2019) and a bit different to what has been described in (Zimmer, Bender, and Pollok 2018). Although the library is approximating the thermodynamic state on the steady mass-flow pressure, this is not made explicit. This means in practice that when one calls the function `Medium.pressure(inlet.state)`, the return value is the steady mass-flow pressure $\hat{p}$ and not p. An explicit denotation of the steady mass-flow pressure simply turns out to be too cumbersome and unpractical on the existing media model base. Yet it is important to have the underlying approximation in mind.

Also a minor extension to the standard Media library is needed to support our interface: a function is added that retrieves the mass fraction for given thermodynamic state. For this reason, the DLR Thermofluid Stream Library currently uses a modified copy of Modelica.Media. Once the standard has been updated, the copy will be removed. Furthermore, XRG Simulation GmbH provided media models of refrigerants and further media that are compatible to our library.

A classic component which has a single stream from inlet to outlet can be build upon the following basemodel that already contains the law for the inertial pressure gradient.

Listing 2. SISO basemodel

```
partial model SISOFlow
  replaceable package Medium;
  parameter Utilities.Units.Inertance L =
      dropOfCommons.L;
  Inlet inlet(redeclare package Medium=
      Medium);
  Outlet outlet(redeclare package Medium=
      Medium);
protected
  outer DropOfCommons dropOfCommons;
equation
  inlet.m_flow + outlet.m_flow = 0;
  der(inlet.m_flow) * L  =  inlet.r -
      outlet.r;
end SISOFlow;
```

Although simplified w.r.t to the actual implementation, the code also displays the use of a global "DropOfCommons" model that is used to describe many generic parameters shared among many components. Among them is a default value for the inertance because oftentimes a replacement assumption is used and the corresponding dynamics is just seen as a way to reach the desired steady-state solution (or steady mass flow to be more precise).

2.3 Initialization

Per default, we use zero to initialize all mass flows. The occuring gradients in inertial pressure then direct the mass-flow rates toward their natural equilibrium. This approach is similar to ramping up a system from rest or plugging it in to a pressure source. We have successfully applied this method to a variety of systems so far and can confirm the applicability of this approach. The modeler needs to put much less thought into the initialization of its system. Nevertheless if desired, also other options are available.

2.4 Handling zero-mass flow

The initialization approach alone stipulates the requirement that each component must be able to compute with zero-mass flow. Even stronger, we demand that each component can handle reverse flow in a well-natured manner. This means that the equations should not (unnecessarily) destabilize the system and if possible represent a physically plausible behavior. Care has been taken that each component fulfills these robustness requirements. This is especially relevant for active components such as compressors and turbines that add or substract power to a stream of fluid. Also heat-exchangers require careful modeling in this respect.

3 Library Overview

3.1 Library structure

The central entity of the library is the thermofluid stream as described by the connector and the previously listed base-class. There will be boundaries for the stream. These are typically inlets and outlets but note that also volume elements represent boundaries from the perspective of a stream. The inlet of a volume is an outlet of the stream and vice versa.

The topology of the architecture is formed by splitters and junctions. Different from the fluid library in the MSL, there are extra components for this purpose and it is not performed using connector equations. Finally, all those components that manipulate the working fluid of a stream are collected under the term processes. Heat exchangers and valves (or similar mechanism) for flow control are moved up to the highest level due to their significance. Hence the structure as in Figure 2 results.

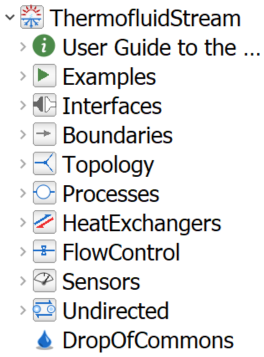

ThermofluidStream
User Guide to the ...
Examples
Interfaces
Boundaries
Topology
Processes
HeatExchangers
FlowControl
Sensors
Undirected
DropOfCommons

Figure 2. Overview of the library and its main packages

Each of the packages contains a sub-package with corresponding test cases. An "Examples" package holds several application examples for the library, that showcase the capabilities and can act as starting points for users. We use both the test cases, as well as the examples, for regression testing during development of the library.

The strategy of the implemented library is that we want to provide robust generic models with only a few parameters for a first iteration of modeling, such as cross- and counter-flow heat exchangers with the ε-NTU method (see Section 3.2.3). These kind of models will be detailed enough to capture the overall system behavior of a wide range of applications and can act as placeholders for higher-fidelity models in later design stages. Since these high-fidelity models will vary for each specific application the modeler will have to implement them for their specific use-case.

3.2 On specific components

3.2.1 Volume models

Volume models take a special role in the thermofluid-stream approach, as they should be used to break algebraic loops for circular fluid flow, isolating the non-linear equation systems locally between the different components (Zimmer 2019a). Each closed fluid loop should contain at least one volume model.

We provide volume models with one or multiple inlets with or without flexible walls, as well as reservoir, accumulator and receiver models, which are also volumes and can be used to break loops. All of these are derived from base classes of volumes making it easy to implement additional specialized volume models if required by the modeler. The base classes define three differential conservation equations $\mathbf{x}_{\text{volume}} = (M, U, M_\text{i})$ with M, U, and M_i, being the mass, inner energy and mass of the different mass fractions in a mixture respectively. For maximum flexibility, volumes offer a broad range of options, like removing inlet or outlet, or adding a heat-port, as well as different initialization methods.

In order to avoid very fast, undampened oscillations between two or more directly coupled volumes or other boundaries, a damping term on the change of mass contained by the volume (Zimmer 2019b) is implemented in all volumes. Contrary to an artificial flow resistance on the inlet or outlet, the damping term does not affect the steady-state solution since it acts only on the change of mass in the volume. Nonetheless, the damping can be switched off by modifying the volumes parameters. With this damping, directly coupled boundaries may still result in very fast oscillations, but at least the dynamics are damped and should be well manageable for a stiff-system solver. If possible though, direct coupling of volumes to other volumes or boundaries should be avoided.

3.2.2 Turbo components

All components that transfer work between a mechanical flange and the fluid share a common partial model "PartialTurboComponent" governed by Equation 5

$$(\Delta p, \tau_\text{s}) = f(\dot{m}, \omega, \Theta_\text{in}) \tag{5a}$$

$$J\dot{\omega} = \tau - \tau_\text{s} \tag{5b}$$

$$\Delta h = \tau_\text{s} * \omega / \dot{m} \tag{5c}$$

where Δp is the pressure gain over the component, ω is the angular velocity, J the moment of inertia, τ the torque applied to the flange, τ_s the torque needed to maintain static operation in the current conditions and Δh the specific enthalpy the fluid gains from inlet to outlet.

Equation 5a represents the pressure/torque characteristic of the component and is not implemented in the partial class. Different implementations of f in child classes allow for pumps, compressors, turbines, fans or other turbo components and for different levels of fidelity. Note that Equation 5 holds no assumption on the specific thermodynamic process of the turbo-component (e.g. compression / expansion with a fixed isentropic coefficient), since different processes can be implemented by different functions f in Equation 5a.

Equation 5b can be replaced by a boundary condition on ω, when the angular dynamics of the component is not of interest.

Equation 5c is additionally normalized for low mass-flow, effectively limiting $|\Delta h|$. If the fluids enthalpy is increased in the component, any work the fluid cannot take on is dumped onto a heat-port. If enthalpy is taken from the fluid (e.g. in a turbine) τ_s is reduced to still fulfill Equation 5c in case of normalization, limiting the work that can be taken out of the fluid.

3.2.3 Heat exchangers

For heat exchange between two fluids, two different approaches are used in this library. For applications with single-phase fluids on both sides, the ε-NTU method is implemented. For fluids with phase transition (for example refrigerants), the heat exchanger is discretized in multiple heat exchanging elements.

ε-NTU method

When the outlet temperatures of the heat exchanger are not known a priori, the ε-NTU method is most convenient. It provides relatively simple correlations for different types of heat exchangers (counter-flow, cross-flow, parallel-flow, etc.). The effectiveness ε of a heat exchanger is defined as the ratio between the actual and the maximum heat flow rate:

$$\varepsilon = \frac{\dot{Q}}{\dot{Q}_\text{max}} = \frac{\dot{Q}}{C_\text{min} \Delta T_\text{max}} \tag{6}$$

To obtain the maximum possible heat flow rate, the heat capacity rates $C = c_\text{p}\dot{m}$ on both sides (hot/cold) of the heat exchanger are compared. The side with the smaller heat capacity rate ($C = C_\text{min}$) needs less energy to experience the maximum temperature difference ($\Delta T_\text{max} = T_\text{h,in} - T_\text{c,in}$).

With those quantities, the so called Number of Transfer Units (NTU) can be calculated:

$$NTU = \frac{kA}{C_\text{min}} \tag{7}$$

where k is the overall heat transfer coefficient and A is the surface area for heat transfer. For any heat exchanger it can be shown that (Schlünder et al. 1997):

$$\varepsilon = f(NTU, C_\text{r}) \tag{8}$$

This means the effectiveness ε of the heat exchanger is a function of the dimensionless number of transfer units and the ratio of the minimal and maximal heat capacity rate $C_r = \frac{C_{min}}{C_{max}}$. For each type of heat exchanger, a specific relation can be found in literature, for example (Incropera et al. 2007). For cross-flow heat exchangers with both fluids unmixed, the effectiveness can be obtained by:

$$\varepsilon = 1 - exp\left[\left(\frac{1}{C_r}\right) NTU^{0.22} exp[-C_r NTU^{0.78}] - 1\right] \quad (9)$$

This correlation and similar ones for counter-flow are implemented in the library.

To determine the thermodynamic state at the outlet of the heat exchanger, the approach is slightly modified to our purpose. To avoid non-linear equation systems, especially when connecting multiple heat exchangers in series, it is beneficial to use the outlet enthalpy as a state. Therefore the efficiency of the heat exchanger is formulated in terms of specific enthalpy:

$$\Delta h = \varepsilon \Delta h_{max} = \varepsilon c_p \Delta T_{max} \quad (10a)$$
$$h_{out} = h_{in} - \Delta h; \quad (10b)$$

And thus the actual heat flow rate can be obtained from:

$$\dot{Q} = \dot{m}\Delta h \quad (11)$$

To become a state variable, the outlet enthalpy h_{out} is filtered with a first order term with time constant τ_{filter}:

$$\frac{\partial h_{out}}{\partial t}\tau_{filter} = h_{in} - \Delta h - h_{out} \quad (12)$$

The thermodynamic state is eventually retrieved from the specific outlet enthalpy h_{out} to prevent the creation of non-linear equation systems.

Discretized Heat Exchanger

When condensation and evaporation become relevant in a heat exchanger, a method is needed that is able to handle phase transition. Generally there are two main approaches suitable for this application: the moving boundary approach and the discretization of the heat exchanger into a finite number of elements. In this library, the latter is implemented because it promises very robust behaviour and can easily be understood.

The discretized heat exchanger consists of N heat conducting elements on each side of the fluid. They are connected via a thermal conductor from the MSL. The number of discretization elements can be set by the modeler. Figure 3 shows how the single elements are connected to each other. The fluid ports are arranged in terms of a counter-flow heat exchanger.

When dividing the heat exchanger in several elements, it is advantageous to model them in a way that no oscillations occur when multiple elements are connected to each other. To this end, the mass in each element is assumed to be quasistationary $(M = \rho V)$ and the inlet mass flow is coupled to the outlet mass flow $(\dot{m}_{in} = -\dot{m}_{out})$. While this assumption neglects the change of enthalpy attributed to dM/dt for changing densities, in our experience it does not change the result drastically while reducing the problem's complexity. Now, the energy balance of each element is stated in the following form:

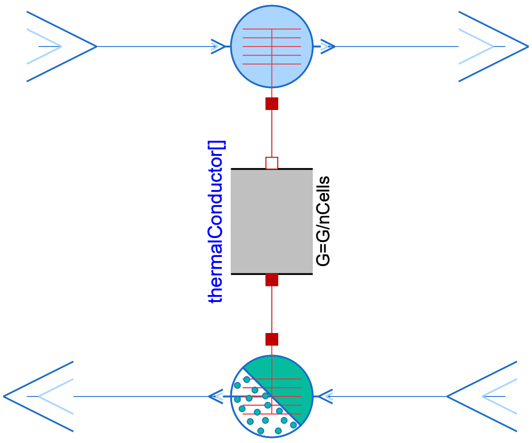


Figure 3. Cell model of discretized heat exchanger

$$M\frac{\partial h}{\partial t} = \dot{Q} + \dot{m}(h_{in} - h) + V\frac{\partial p}{\partial t} \quad (13)$$

where h is the specific enthalpy at the outlet of the element. For the sake of robustness, the change of pressure in the fluid is neglected in the energy equation (13) $(V\frac{\partial p}{\partial t} = 0)$ and therefore the pressure input is not required to be smooth.

The convective heat transfer from or to the fluid is calculated as follows:

$$\dot{Q} = UA(T_{heatPort} - T_{surface}) \quad (14)$$

where U is the coefficient of heat transfer and A the surface area. In general, a detailed calculation of the coefficient of heat transfer is not trivial. Hence we offer a pragmatic approach exploiting that the coefficient of heat transfer U can be stated in terms of the Nusselt-Number (Incropera et al. 2007) which in turn depends on the Reynolds number (presuming forced convection):

$$Nu = \frac{UL}{\lambda} = CRe^m Pr^n \quad (15)$$

with the characteristic length L and the conductivity of the fluid λ. The values of the coefficient C and the exponents m and n are dependent on geometry and flow characteristics. Since the Reynolds number is proportional to the mass flow $\dot{m}$, we can derive a simple scaling law based on the Reynolds exponent m:

$$U = U_{nom}\left(\frac{|\dot{m}|}{\dot{m}_{nom}}\right)^m \quad (16)$$

For turbulent flow the Reynolds exponent m is equal to 0.8. The coefficient of heat transfer for the multiphase element has to be estimated differently. It is determined by the actual phase in each discretization element, therefore it is dependent on the vapor quality. According to the single phase, for each phase (liquid, vapor, two-phase) a nominal coefficient of heat transfer can be set and the actual coefficient is calculated accordingly:

$$U_{\text{liq}} = U_{\text{liq,nom}} \left(\frac{|\dot{m}|}{\dot{m}_{\text{nom}}} \right)^{m_c} \tag{17a}$$

$$U_{\text{vap}} = U_{\text{vap,nom}} \left(\frac{|\dot{m}|}{\dot{m}_{\text{nom}}} \right)^{m_e} \tag{17b}$$

$$U_{\text{tp}} = U_{\text{tp,nom}} \tag{17c}$$

The Reynolds exponents for normalisation of the heat transfer coefficient for evaporation ($m_e = 0.5$) and condensation ($m_c = 0.4$) are taken from (Yan and Lin 1999b) and (Yan and Lin 1999a). In the two-phase region, a constant coefficient of heat transfer is assumed. Furthermore a minimum value for the coefficient of heat transfer U_{min} is introduced to ensure heat transfer at zero mass flow. The coefficient of heat transfer on the two-phase side of the heat exchanger depends on the actual phase. Therefore the vapor quality χ has to be calculated in each element, using the dew and bubble enthalpies of the fluid:

$$\chi = \frac{h - h_{\text{bubble}}}{h_{\text{dew}} - h_{\text{bubble}}} \tag{18}$$

The coefficient of heat transfer used in the two-phase elements thus is formulated as a function of the vapor quality $U(\chi)$. For smooth transition between different coefficients during phase change, an interpolation is applied. The definition of the vapor quality (Equation 18) allows it to go below zero (when subcooled) and above one (when superheated). This allows the interpolation to be formulated across the phase boundaries and avoids jumping in those critical regions. The test models for a condenser and evaporator show robust and valid behavior of the discretized heat exchanger, although the performance is highly dependent on the number of discrete elements. Section 3.3 contains a corresponding application example.

3.2.4 Valve models

Valve characteristics can be very important for the control design and the control authority. The library hence features different types with different pressure gradient curves. Also some functional valve models are directly combined with splitter models in order to facilitate topological changes in complex architectures (Figure 1).

3.2.5 Sensor models

All implemented sensor models can be used in two ways. Firstly, they output the measured signal as a RealOutput as it is common for Modelica sensor models. Additionally, they display the current signal value during simulation using the DynamicSelect command (see Figure 4). This enables the user to get a fast, intuitive understanding of the current state of the simulation without the need for displaying any signal curve. We found the second use very practical and are using the sensors mostly in this way, which is why the actual signal output is conditionally removed by default.

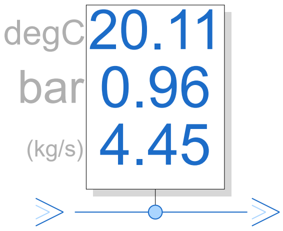


Figure 4. Exemplary sensor displaying three quantities

3.3 Specific solution for undirected flows

Within the library we provide a package containing components that can have undirected flows. It is similarily structured to the main library but contains less components. For these components however, the direction of mass-flow does not need to be known a-priori and is determined dynamically during the simulation. The approach for undirected stream-dominated flow simulation was introduced in (Zimmer 2019a). Note that this approach is still stream-dominated and, while we continue to demand robustness for low and zero mass-flow, the results may not be valid for these conditions. The undirected simulations are therefore interesting for applications where non-zero mass-flow operating points are present for both flow directions while the switching dynamics between mass-flow directions are not of interest. In practice, this will be the case for many applications, like a combined vapor-cycle/heat-pump.

When the direction of mass-flow is reversed, the flow of information is reversed with it. Therefore a undirected connector carries information about the thermodynamic state in both directions: forward and rearward (see Listing 3).

Listing 3. Two connectors for undirected flows

```
connector Thermalplug_fore
  replaceable package Medium;
  SI.Pressure r;
  flow SI.MassFlowRate m_flow;
  input Medium.State state_rearwards;
  output Medium.State state_forewards;
end thermalplug_fore;

connector Thermalplug_rear
  replaceable package Medium;
  SI.Pressure r;
  flow SI.MassFlowRate m_flow;
  input Medium.State state_forewards;
  output Medium.State state_rearwards;
end thermalplug_rear;
```

Each component also computes its influence on the state in both the forward and backward direction (see Equation 19).

$$
\begin{aligned}
\Theta_{\mathrm{out,fw}} &= g_{\mathrm{fw}}(\Theta_{\mathrm{in,fw}}, \dot{m}, \mathbf{x}) \\
\Theta_{\mathrm{out,bw}} &= g_{\mathrm{bw}}(\Theta_{\mathrm{in,bw}}, \dot{m}, \mathbf{x})
\end{aligned} \tag{19}
$$

When connecting these undirectional components in a complex topology, junctions and splitters cannot be distinguished and become generic nodes. The implementation of a node has to avoid cyclic dependencies of the signal flow as well as to provide a regularization around the zero-mass flow regime. It is thus almost a re-implementation of the Modelica stream connector (Franke, Casella, Otter, et al. 2009). Unfortunately the regularization of the stream connector semantics is numerically vague and also not subject to a regularization parameter. The width of the regularization scheme may however play an important role in situations of flow reversal and also influences the eigenvalues of the system. Hence a proper option for parametrization is needed and the Modelica stream connector itself could thus not be used for this library. Here, the default value of the regularization width is specified by the dropOfCommons.

It is strongly advisable to use undirected components only when the flow-direction is really unknown. Knowing the direction a priori is a too valuable piece of information to throw away. For instance, using directed components helps avoiding spurious loops (those which appear in the connection graph but are never realized by the fluid flow) and the need to cut those loops. Also creating models that work in both directions is not always reasonable and certainly creates often code that is needlessly complex. Please note that we provide also adapters between directed and undirected stream networks. These can be used to isolate parts of the network that require undirected flow, and keep the rest directed.

An example architecture that contains undirected components is given in Figure 5. It shows a reversible heatpump as it can be used for residential air conditioning. The specialty of this system is that the direction of the refrigerant flow can be reversed. The heat exchangers can thus act as evaporator or condenser according to the current cycle operation. In cooling mode (blue arrows), the indoor unit acts as an evaporator and the outdoor unit (blue) acts as a condenser. Thus the heat is absorbed from the inside air and rejected to the outside. In heating mode (red arrows) the cycle is reversed which makes the indoor unit the condenser and the outdoor unit the evaporator. Hence the heat is absorbed from the outside and rejected to the inside. The system is built out of a combination of undirected and directed components. It consists of a undirected phase separator (receiver) and two separate metering devices (magenta). This allows us to control the superheating temperature after the evaporator in both operating modes. In practice, the change of flow direction is carried out by a reversing valve. In our example we control the flow direction by a system of valves and undirected junctions (yellow). The example simulates robustly with fixed boundary conditions for the air side and the compressor and the cycle can be reversed during simulation.

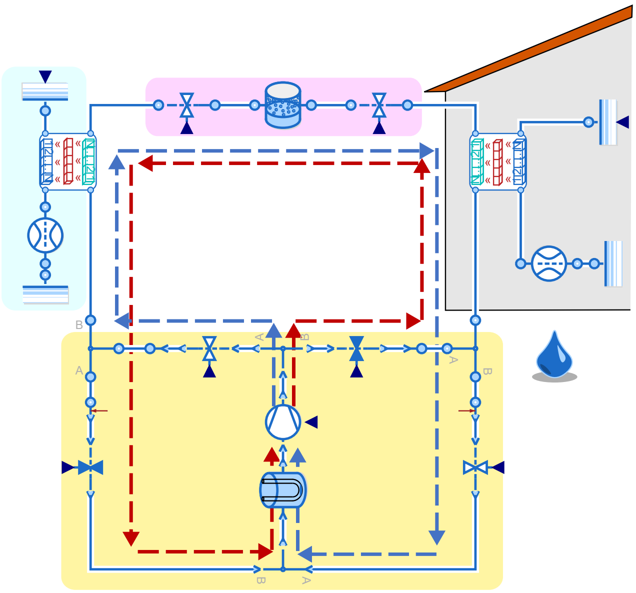


Figure 5. Example of a reversible heatpump that contains two undirected heat exchangers. The operating mode of the cycle can be switched during simulation.

3.4 Specific solution for dynamic pressure

The dynamic pressure arises from the macroscopic motion of the fluid. Its computation for a given mass flow rate hence requires also information on the geometry. Typically the cross-section area is used to compute the velocity:

$$v = \frac{\dot{m}}{A\rho}$$

The dynamic pressure q is then

$$q = \rho/2v^2 = \frac{\dot{m}^2}{2\rho A^2}$$

If the dynamic pressure is vital for each part of the system, it is a natural choice to include the cross-section area in the connector and specify the geometry at each component. However, we do not think that this represents the majority of use-cases for our library. Many thermodynamic processes are adequately described without needing the dynamic pressure. Its occurrence is mostly confined to special sub-systems like ram-air inlets, venturi pumps, or diffusors. Under this presumption, the need to describe the geometry in all components does more harm than good. Providing a localized solution for the dynamic pressure seems to be the most useful approach. In this way, it can be considered when needed and ignored otherwise. Hence, we introduce special boundaries for entering and exiting a zone where dynamic pressure is considered. One side of this boundary expresses the static pressure for a specified velocity assumption whereas the other side computes the velocity resulting from mass flow-rate, density and cross-section area. The boundary then assigns a pressure difference so that the total pressure balance is upheld. A typical use of such a boundary is its use of ram-air going through a heat exchanger as in a car as shown in Figure 1.

A typical component that uses dynamic pressure is a nozzle that accelerates (or decelerates) a fluid. In case the fluid is diffused an increase in mass-flow rate will increase the outlet pressure. Hence a correct formulation of the boundary conditions is needed because otherwise the inertial dynamics of the fluid may destabilize the system. A corresponding example of the Venturi-effect is part of the library.

4 Easy-to-read, easy-to-adapt

4.1 General modeling style of components

In general, the library is structured by a flat hierarchy, limiting the use of partial models, functions and packages to only those cases with high usefulness. This is done intentionally to increase the readability and understandability of the individual components while still utilizing the might of object-oriented modeling when it is of benefit.

Components are implemented to provide a formulation of Equation 1 in explicit form, whenever feasible. Nonetheless, sometimes certain quantities that are required to compute the outlet state Θ_{out} depend not only on the inlet state Θ_{in} mass-flow $\dot{m}$ and component state $\mathbf{x}$, but also on the outlet state itself. An example is the change in dynamic pressure, that depends on the outlet density. This typically results in systems of non-linear equations within the component. While the stream-dominated approach still manages to keep the non-linear systems separated in small local ones, for general simulation speed and the real-time application of the library, even small non-linear systems are undesirable. Therefore we decided to avoid non-linear equation systems larger than size 1. This can be achieved by reformulating the equations in a manner and/or applying simplifications until the components' non-linear equation systems drop to the desired size, or by introducing the problematic quantity as an artificial state within the component $\tilde{x}$, effectively implementing a fast low-pass on it, as it was done in subsubsection 3.2.3. While the latter solution avoids simplifications, it introduces additional artificial states, whose time-constants have to be chosen with care. While this is not an ideal solution, when done right the result is a fast running and accurate simulation, with only local non-linear systems of maximum size 1. For hard real-time applications the remaining non-linear systems can still be resolved by adding additional states in the corresponding components.

While implementing the models, we documented the sources of equations, as well as the simplifications and mental models for the components in code and the documentation annotation. Furthermore, intuitive icons provide a good readability of the high-level models.

4.2 Adapt to your own needs: A use case

While many use-cases can be simulated by the libraries standard components, many others will require specialized components. The relative flat implementation style enables the modeler to easily implement own components and adapt the library for their needs. An example for this is provided in Figure 6. It depicts a simulation model for an espresso machine.

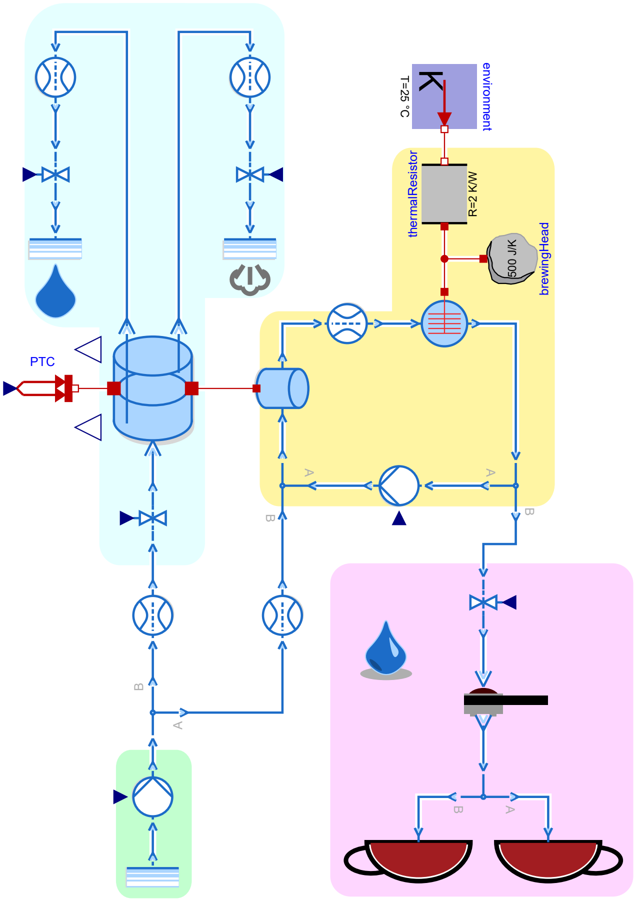


Figure 6. Example of a model with individualized component models. This model simulates an espresso machine composed of water source and pump (green), boiler, water and steam outlets (blue), brewing head with cycling water (yellow) and valve, coffee strainer and cups (magenta). Cold water enters the machine through a source at atmospheric pressure and gets pressurized by a pump to 8.5 bar. Part of the water goes through a control valve into a boiler, where the pressure is regulated by heating through a PTC and the water level is controlled by the valve. For this purpose the boiler outputs both of these quantities. From the boiler, steam or boiling water can exit if the corresponding valve is opened. The boiler is in thermal contact to a secondary small volume from which water cycles to the brewing head and back, heating up the brewing head. If the machine is at temperature, the valve in front of the coffee strainer is opened and water flows from the pump into the cups, picking up heat in the secondary volume and the brewing head.

While many standard components are used, the sinks for steam and boiling water, the coffee strainer, the cups and the boiler are specialized components. All except the boiler are internally composed of standard components and only implement a specialized icon. The coffee strainer consists of a flow resistance and each cup contains a flow resistance and a sink, as well as a variable to track the level of coffee in the cup. The two sinks for steam and water only update the image of the original sink.

Contrastingly, the boiler, while derived from the standard volume, is a fully individual component. It has one inlet and implements the conservation equations for mass and energy similar to a normal volume, but contains two heat-ports and two specialized outlets with states on the bubble- and dew-line respectively, instead of one normal outlet. Additionally, liquid level and pressure are output as real numbers and a different icon is implemented.

5 Concluding remarks

5.1 Library source and terms of use

The library is available on GitHub:
`github.com/DLR-SR/ThermofluidStream`

It is available under the 3-Clause BSD License. It has been developed using Dymola and is based on Modelica 3.2.3. Pendantic checking has been applied to all components in order to improve cross-tool compatibility. Compatibility with Open Modelica and Modelon Impact has been achieved to a large degree and is documented in corresponding issues on GitHub.

If you publish work that is based on this library, please cite this paper and (Zimmer 2020). We also welcome feedback in form of issues raised on GitHub. Also when you have positive feedback, you can feel free to raise an issue to share your experience. Have fun!

5.2 Future Work

We will keep maintaining this library and improving its components. Apart from having robust models, hard realtime capability, suitable control design as well as various forms of health monitoring are our main research and development interests.

Acknowledgements

This work has been partially supported by the Helmholtz Gemeinschaft in frame of the research project for onboard diagnosis. Hence we would also like to thank the team of Robert Bosch GmbH for feedback and motivating use-case scenarios. Further thanks goes to Chistopher Laughman from Mitsubishi Electronic Research Center for providing encouraging feedback. Also the work of Tobias Krenz form TU Darmstadt offered a valuable perspective. We are still grateful to Airbus and our former colleagues Dr. Alexander Pollok and Dr. Daniel Bender who were pioneering the new methodology. Also we like to thank our colleague Dr. Peter Eschenbacher for his valuable modeling experience on valves and pumps. Thankfully, the Open Modelica team supported us with respect to tool compatibility and Stefan Wischhusen from XRG generously provided additional media models.

References

Casella, Francesco (2005). “Object-Oriented Modelling & Simulation of Power Plants with Modelica”. In: *Proceedings of the 44th IEEE Conference on Decision and Control* (Seville, Spain), pp. 7597–7602.

Casella, Francesco et al. (2006). “The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks”. In: *Proc. 5th International Modelica Conference* (Vienna, Austria), pp. 559–568.

EcosimPro (2021-08-03). *FLUIDAPRO*. URL: https://www.ecosimpro.com/products/fluidapro/.

El Hefni, Baligh and Daniel Bouskela (2014). “Dynamic modelling of a Condenser with the Thermo SysPro Library”. In: *Proceedings of the 10th International ModelicaConference* (Lund, Sweden), pp. 1113–1122.

Franke, Rüdiger, Francesco Casella, Martin Otter, et al. (2009). “Stream Connectors – An Extension of Modelica for Device-Oriented Modeling of Convective Transport Phenomena”. In: *Proceedings of the 7th Modelica Conference* (Como, Italy), pp. 108–121.

Franke, Rüdiger, Francesco Casella, Michael Sielemann, et al. (2009). “Standardization of Thermo-Fluid Modeling in Modelica.Fluid”. In: *Proceedings of the 7th Modelica Conference* (Como, Italy), pp. 122–131.

Incropera, Frank et al. (2007). *Fundamentals of heat and mass transfer*. Wiley New York.

Otter, Martin et al. (2019). “Thermodynamic Property and Fluid Modeling with Modern Programming Language Constructs”. In: *Proceedings of the 13th International Modelica Conference* (Regensburg, Germany).

Process Systems Enterprise (2021-08-03). *gPROMS*. URL: www.psenterprise.com/products/gproms.

Schlünder, Ernst-Ulrich et al. (1997). *VDI-Wärmeatlas: Berechnungsblätter für den Wärmeübergang*. 8. überarb. und erw. Aufl. Springer.

Sielemann, Michael et al. (2011). “Optimization of an unconventional environmental control system architecture”. In: *SAE International Journal of Aerospace* 4.2.

Yan, Yi-Yie and Tsing-Fa Lin (1999a). “Condensation heat transfer and pressure drop of refrigerant R-134a in a small pipe”. In: *International journal of heat and mass transfer* 42.4, pp. 697–708.

Yan, Yi-Yie and Tsing-Fa Lin (1999b). “Evaporation heat transfer and pressure drop of refrigerant R-134a in a plate heat exchanger”. In: *Journal of Heat Transfer* 121.1.

Zimmer, Dirk (2019a). “Robust Simulation of Stream-Dominated Thermo-Fluid Systems: From Unidirectional to Bidirectional Applications”. In: *EUROSIM Congress 2019* (Logroño, Spain).

Zimmer, Dirk (2019b). “Towards hard real-time simulation of complex fluid networks”. In: *Proceedings of the 13th International Modelica Conference* (Regensburg, Germany), pp. 579–587.

Zimmer, Dirk (2020). “Robust object-oriented formulation of directed thermofluid stream networks”. In: *Mathematical and Computer Modelling of Dynamical Systems* 26.3, pp. 204–233.

Zimmer, Dirk, Daniel Bender, and Alexander Pollok (2018). “Robust Modeling of Directed Thermofluid Flows in Complex Networks”. In: *Proceedings of the 2nd Japanese Modelica Conference* (Tokyo, Japan), pp. 39–48.

The Potential of FMI for the Development of Digital Twins for Large Modular Multi-Domain Systems

Marcus Wiens[1] Tobias Meyer[1] Philipp Thomas[1]

[1]System Technology, Fraunhofer IWES, Germany,
{marcus.wiens,tobias.meyer,philipp.thomas}
@iwes.fraunhofer.de

Abstract

Digital twins enable the observation, prediction, and optimization of a physical system and thus allow to realize their full potential. However, their functionality is mainly based on simulation models of the entire system behavior. For modular multi-domain systems, this requires the extensive use of dynamically composed models that are made up of individual component models. The FMI-Standard forms a solid foundation for this problem and is very well known in the automotive engineering fields. However, composed system models using FMI are not widely adapted in renewable energy and wind energy yet. So far, the coupling of simulation models is limited. This paper discusses the strategy of building digital twins from individual FMUs with predefined model interfaces based on an ontology for renewable energy systems. An accelerated development is enabled by the exchange of sub-models in the digital twin without adjustments of interface. An example for the proposed process is given by the composed simulation model of a hydrogen generation process based on wind energy.

Keywords: Digital Twin, Functional Mockup Unit, Wind Energy, Renewable Energy, Ontology

1 Digital Twins in Wind Energy

Digital Twins are a major contribution for the digitalization of physical systems and processes. The digitalization of systems is a key part of Industry 4.0, which focuses on the development of cyber physical systems. As a result, a digital twin enables use-cases e.g., continuous observation, prediction, optimization of logistics etc., to maximize the potential of physical systems (Tao et al.2019). There are multiple definitions of digital twins, like the often-cited definition from NASA: *“The Digital Twin is an integrated multiphysics, multiscale, probabilistic simulation of an as-built vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its corresponding flying twin.”* (Glaessgen and Stargel. 2012) and many more (Fuller et al.2020). Common to them is that there is a digital entity representing a physical entity and the adaptation of both objects based on an exchange of data. To avoid misconception of digital twins, they are categorized (Kritzinger et al.2018) into distinct definitions by their level of integration:

- Digital Model: A digital copy of an existing or planned physical entity. There is no automated exchange of data between the digital copy and the physical entity. The data of the digital copy might be used in the development process of the physical entity, but a change in the digital copy has no immediate effect on the physical entity.
- Digital Shadow: A digital shadow extends the digital copy by an automated one-way exchange of data. The physical entity changes the state of the digital shadow, but not in the other way.
- Digital Twin: The digital twin has full level of integration, which is realized by the automated exchange of data between the physical entity and the digital copy. Both systems affect each other.

Those categories build on top of each other and can be interpreted as development steps of a digital twin. In fact, the development of a digital twin not only consist of the creation of the digital entity, but also of the surrounding elements for data exchange and interfaces for services. The Generic Digital Twin Architecture (Steindl et al., 2020) sets a well-defined structure for all elements of a digital twin. Furthermore, the use of Ontologies is demonstrated to organize the services and internal functionality of the digital twin. Ontologies give structure to information and make them processable by a machine (Gruber. 2016).

In wind energy, digital twins are primarily developed for components or very large systems. Among large-scale projects, Ramboll's "True digital Twin" (Tygesen et al. 2018), which represents offshore structures, is particularly noteworthy. At the component level, the drive train is represented in detail, for example by Winergy (Flender International GmbH) or Schaeffler (Schaeffler AG). However, they are either designed specifically for the capabilities of a specific product or are limited to the suppliers’ own components. The development of holistic full-turbine digital twins is currently at the stage of building the virtual entities, without the required surroundings. The wind turbine model adjustment for an existing real wind turbine (Pimenta et al.2020) shows potential for a virtual entity, but the process requires many steps. Another method (Branlard et al.2020) presents the use of linearized models in combination with a Kalman Filter to estimate real-time load and fatigue. The applications of the presented digital twins lie in the tracking of fatigue damage or evaluation of alternative operation strategies. However, those systems implement single component (one wind turbine model) digital twins, which are not designed for further coupling of simulation models. We assume that a coupled model will be necessary for the development of digital twins for large systems like an entire wind farm or green hydrogen production facilities.

Furthermore, current challenges in research of digital twins are stated in a review paper (Fuller et al.2020) for the fields of manufacturing, healthcare and smart cities, but their findings seem to be more general according to other papers (Tao et al.2019):

- Unified development framework
- Lack of clear definitions
- Standardization
- Lack of large-scale projects due to missing domain knowledge

Hence, the development of digital twins is in an early stage. This article addresses the above issues by proposing a development process for the digital model of the physical entity based on Functional Mock-up Interface (Modelica Association Project FMI) (FMI) and Co-Simulation. A Co-Simulation is the technique of the simulation of a coupled system based on the composition of single simulators (Gomes et al.). Multiple abstraction levels (van Nguyen et al.2017) have to be considered.

2 Digital Twins for Large Modular Multi-Domain Systems

Large multi-domain systems are composed modularly of individual components. Each component can be changed individually, e.g., by means of a reconfiguration of its behavior, by a change in physical properties or be a full exchange of the entire component. The entire system can be reconfigured by removing or adding components. All these aspects need to be reflected in the simulation models. The large number of possible configurations makes it impossible to cover them all with a dedicated full-system simulation model. Instead, a similar modular approach for the system model is required. Additional complexity arises, since systems with a large scope usually cover multiple domains of different engineering fields. This usually requires the combination of models from different modelling tools.

The modular setup of a full-system model from individual component models is a prime candidate for the assembly of a full-system model from individual Functional Mock-up Units (FMUs). FMUs allow simulation models from various modeling tools to be coupled. The simulation models are contained as a binary library file and the underlying algorithms remain hidden. Two types of FMU are specified in the FMI standard: co-simulation and model exchange. Only co-simulation FMUs are used here. In addition to the simulation model, co-simulation FMUs also contain the simulation solver. As a result, the FMU independently computes simulation results for each simulation time step and does not require a higher-level solver. In this way, models with different integrator time steps can be coupled, for example the mechanical system of the wind turbine (time step: 10 ms) and the electrical system (1 ms). This make FMUs an ideal base for digital twins of large multi-domain systems.

The process is most useful if a model database in combination with a dedicated full-system model assembly tool is used. This requires predefined interfaces and multi-dimensional connectors between single models for an efficient modelling. Predefined interfaces are needed, since large scale systems of multiple domains have a various number of elements and interconnections and therefore can make the combination of models complicated between domains. An ontology is one possible solution to this problem. In fact, the ontology can manage the knowledge about model interfaces and thereby be an instruction for in- and outputs for specific objects. This implies the implementation of specific adapters based on the ontology in the utilized modelling tools. Additionally, the ontology should give unique names for connections of the same kind. This eases the (automated) connection later. An example for an ontology is given in Section 3. Already existing models (e.g., engineering models), which are considered for the model database, can be connected with those adapters in their specific modelling tools. In this way, there is no coupling between the model in- and output and the required connections from the ontology. In the end the models are exported as FMU to the database.

The usage of an ontology for the definition of abstract interfaces yields the additional advantage that FMUs, which model components in different manners, can be exchanged without additional changes to the full-system model. In our digital twin approach, it is possible to exchange a detailed simulation model to a coarser simulation model or vice versa since the model interface remains unchanged. This allows for a high degree of flexibility when adapting to the requirements of different usage scenarios. Furthermore, the model database benefits from predefined "real world" interfaces to connect the digital models with the physical entity. This is represented by an FMU model which maps measurement data streams

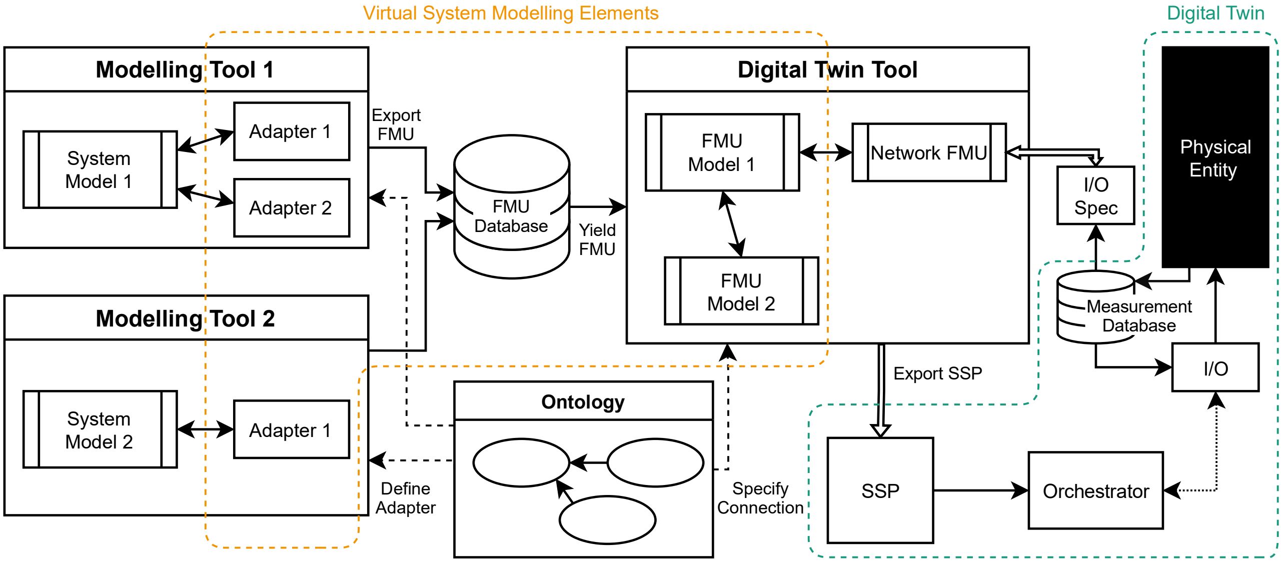


Figure 1 Visual representation of the development of digital twins for large modular multi-domain systems

or parameter data streams from the physical system to FMI input/outputs.

The implementation of multi-dimensional connectors that represent physical connections allows for easier handling of models with in- and output of high dimensions. Currently, in and output in FMUs are one dimensional and for high dimensional signals this results in hardly manageable input and output enumeration. While this drawback will be solved with FMI 3.0, wide adaptation in multi-purpose simulation tools, which are used to create FMUs, cannot be expected soon.

Finally, this leads to the idea of a dedicated development framework, which is visualized in Figure 1 along with the proposed modelling process. The framework accesses the database and eases the building of the digital twin based on single elements and real-world interfaces, similar to the “Open Simulation Platform” (DNV GL AS) for engineering with co-simulations. The adapters represent intuitive connectors which are based on the ontology: Connections between individual models are specified by the user on the top level and details are handled by the framework. A main differentiation between our approach and the open simulation platform is our desired full automation of the model assembly process. Instead of user-built models, the full-system model is derived from the ontology-stored structure information automatically. This creates further requirements for the tools used to assemble individual models into the full-system model. Furthermore, the use of co-simulation FMUs decouples the modeling and numerical solution of the individual simulation model (within one FMU) on the one hand from the simulation of the coupled full-system model on the other hand. The coupling of FMUs is described by the System Structure and Parameterization (SSP) Standard, which builds the output of the model assembler. The SSP and FMU files are imported by an Orchestrator to run the virtual side of the digital twin.

Eventually, this procedure can be integrated into the development phase of a new product. The sensor layout can be optimized along with the product and a digital twin can be delivered faster, which is a step forward in the digitalization of renewable and especially wind energy.

2.1 General Process Description

We propose the following process for the structured development of model-based digital twins. The process can be separated into two distinct main steps, assuming the ontology is already defined beforehand. First, an FMU model database is created:

- Creating a simulation model: Use existing engineering models or develop new model. Use model reduction techniques to achieve real time capability, if necessary
- Define required adapter connection models in the specific modelling tool based on an ontology
- Connect the model to the corresponding adapter
- Export to FMU and upload into the database

This builds the base for development of digital models. The next step is the combination of single models into a complex system:

- Collect all components for the full-system multi-domain digital twin
- Connect single FMU models with a toolbox (see Section 2.2 for an example)
- Add the communication adapters for the digital twin

- Create SSP package with all FMU models and connection description
- Run Simulation using an orchestrator e.g., FMPy

The second step is basically the creation of the virtual entity for the digital twin. A complex virtual entity can be build based on component models, which are connected by intuitive adapters. The communication adapters take the responsibility of steering signals from inside the model to the outside for the digital twin.

2.2 The fmuToolbox

For ease of automated simulation, we implemented a Toolbox in Python for the setup of Co-Simulations based on SSP. Since the SSP file is written in an XML-Format, we use the `lxml package` in Python for file processing. The toolbox consists of multiple functions which represent the single steps of the Co-Simulation setup:

- XML-Description: Create a SSP with the component tag of one FMU. The information is extracted from the FMU with FMPy and written according to SSP
- Combination: Collect single XML description files into one SSP by listing all component tags in one file
- AutoConnect: Analyze the inputs-outputs and create connections tags for matching input-output pairs
- Packaging: All FMU files and the SSP description are packed together into an archive file format

Finally, the archive file can be simulated with FMPy orchestrator. The AutoConnect feature requires a standardized naming pattern for the in- and outputs. This can be built e.g., based on a common ontology.

3 Modelling Example: Hydrogen Electrolysis with Wind Energy

The setup of the full-system simulation model for the digital twin of a wind-to-hydrogen facility serves as modeling example. It is considered as a multi-domain system, as the domains wind energy and hydrogen generation are interacting. A digital twin of a wind-to-hydrogen facility would allow for operational optimization and error detection. Therefore, the system behavior needs to be modeled as closely as possible.

In this example, we concentrated on the following components: an aero-elastic wind turbine, wind turbine controller, generator, grid, transformer, converter, and electrolyser to build the wind-to-hydrogen facility. Separate models for the individual plant components exist in multiple simulation environments and are exported as FMUs. An ontology is defined for efficiently modelling and connecting all models from different tools and presumably different developers. The ontology is modelled according to the requirements of the specific use case. All implemented knowledge in the ontology has a specific purpose.

The exemplary ontology is based on the RDF Schema (Brickley et al.2014) to build upon the idea of classes, properties, and triplets. Our ontology defines the additional resources: “UseCase”, “Component”, “Connector”, “Signal”, “Measurement” and “Direction”. Classes and instances, which define the connection between components, are shown in Figure 2 (Remark: All visualizations of the ontology show a fraction of the single overall ontology). The ontology contains the definition of “use cases” to represent a specific simulation setup. An instance of “UseCase” like “Simulation_1” stores information about the instances of components by the property “:simulates”. Instances of any classes are identified by “rdf:type”. Only an instance of a wind

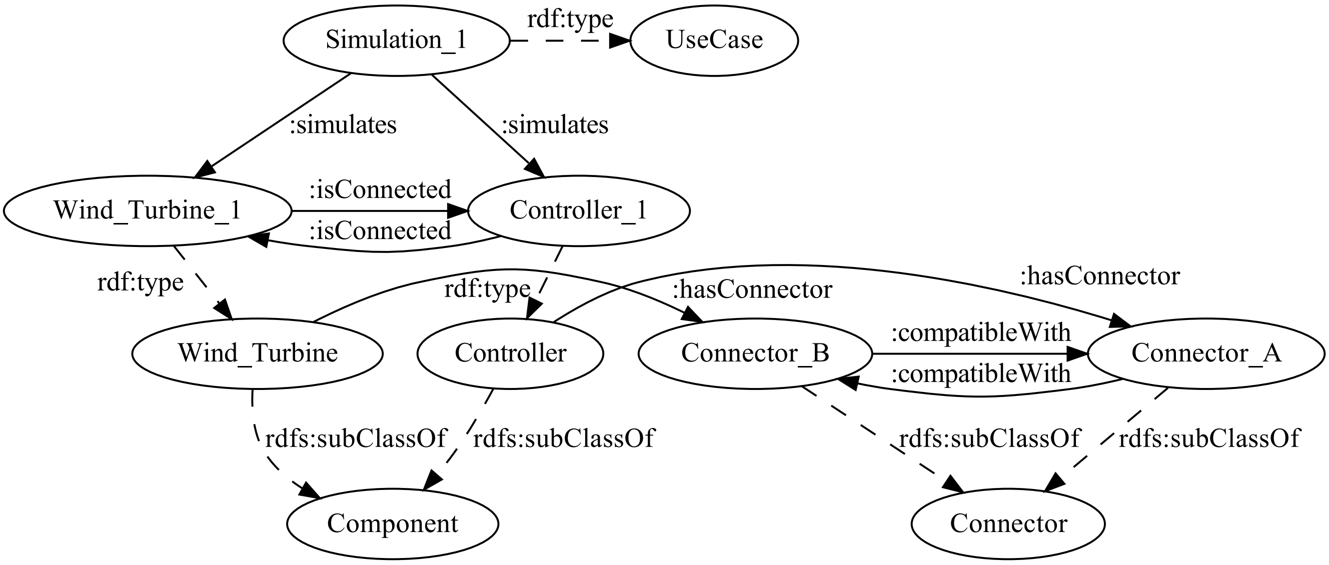


Figure 2 Example for the system ontology and interaction of parts

turbine, controller and their connectors are shown here to ensure clarity. The other components would be added to the ontology in the same manner. Instances of subclasses "Component" are defined to be simulated in a "UseCase" instance. Additionally, they can have any number of instances of subclasses of "Connector". In the case of the component "Wind_Turbine" the only connector is "Connector_B", which is compatible with "Connector_A". Only when two subclasses of "Connector" are defined to be compatible with each other, then a valid connection between instances of "Component" can be made. The same applies for the "Controller" subclass and the corresponding connector instance. This, also shown in Figure 2, helps a modelling software to build valid connections.

The connectors are explained here for a simplified version of the controller connector. An example for the sub-ontology is shown in Figure 3.

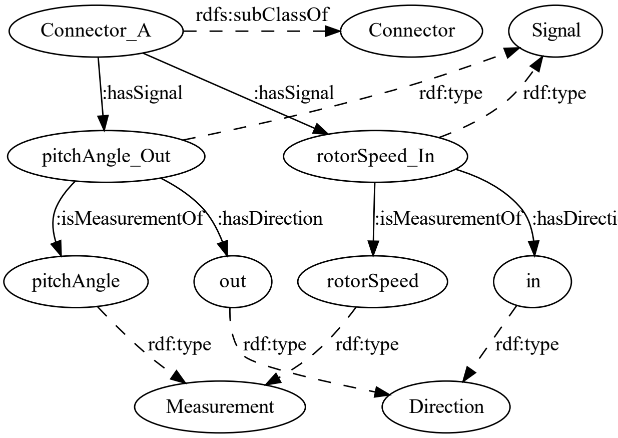


Figure 3 Example ontology for the definition of connectors

Instances of "Connector" have a list of instances of "Signal". They are defined by their measurement and direction. A compatible connector would be built by inverting the direction of all signals in a connector. Names of input or outputs of FMUs can be derived from the ontology. The components and their required signals in this example are added to the full ontology in the described manner. Overall, all this information in the ontology is needed to create file according to SSP definition for the simulation with an orchestrator.

Moreover, specialized ontology tools, e.g., TopBraid Composer, enable to make a query to the ontology with SPARQL. A query as shown in Listing 1 is performed for the component class (Wind_Turbine) to get a list of its connectors. An in- and output list is obtained from the description of the found connectors, which define the required adapter model (as defined in Section 2).

Listing 1. SPARQL query example

```
DESCRIBE ?connector {
   ?subject rdfs:subClassOf :Component .
   ?subject :hasConnector ?connector
   FILTER(?subject = :Wind_Turbine)
}
```

After the ontology is fully defined, the models of the single components are of concern. The wind turbine has rated power of 8 MW and is modelled in MoWiT (Thomas et al. 2014) which is simulated with Dymola. Along with the wind turbine model, the controller is accessed through Dymola as well. For controller development, Matlab/Simulink is used. The Bladed-compatible DLL format is a widely adopted standard in wind energy for controller exchange. The Simulink model is compiled accordingly as DLL and tied into MoWiT using dedicated Modelica-code. The other models are created with Simulink and are directly exported as FMU with respect to the input and output names based on the ontology. All electrical components model the current flow, where the grid consumes all excessive energy. Lastly, the Electrolyser, which is an upscaled model (Espinosa-López et al.2018), uses up 1 MW for hydrogen production. These engineering models were validated in several research projects. The connection between the components is shown in Figure 4.

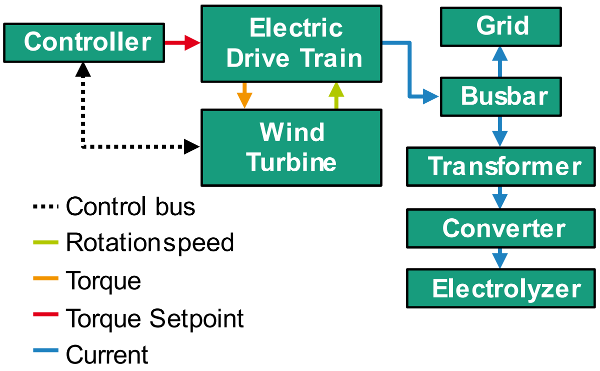


Figure 4 Connectivity diagram of the electrolysis process model

This model contains couplings in the drivetrain between the mechanical rotation and electromechanical torque. Both are controlled by the controller. The control exchanges a lot of data with the wind turbine (~ 40 connections), which are connected efficiently with the AutoConnect feature of the described toolbox. Furthermore, the electric drive train model has an energy output. The energy is distributed by current in the busbar to the grid and electrolyser. The transformer adjusts the voltage level, and the converter changes the current from AC to DC. In the end, the electrolyser consumes energy and produces hydrogen.

A demonstration of the simulation results is shown in Figure 5. The specific simulation case represents the startup of the wind turbine at constant inflow conditions. First, the wind turbine accelerates to enable power output and then continues to steady state operation. The electrolyser consumes the generated power. This process can be used to monitor the energy flow and detect errors when the physical model changes with respect to the digital model. To serve as digital twin for the real plant, the simulation model requires additional connections to take operational real-time data into account and to provide a feedback into the plant operation. A full setup of a digital twin is not within the scope of this paper. The next step is to extend the presented process of building a digital twin full-simulation model onto operational data.

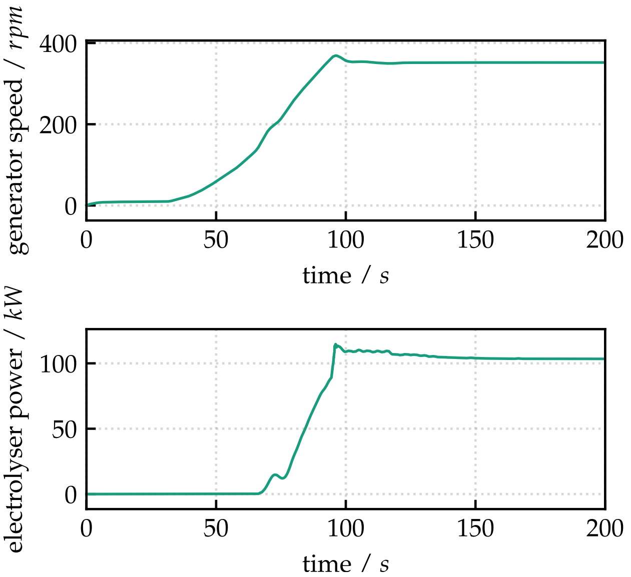


Figure 5 Simulation result for constant inflow condititons

4 Conclusion

In renewable energy and in particular in wind energy, highly modular systems require highly modular simulation models as well. Individual component models to exist from component engineering but linking them to a full-system model requires the efficient use of co-simulations using FMI. More standardization is needed in the development process of engineering models, to enable a fast connection process in the setup of the Co-Simulation. Ontologies offer a formalized way of the interface and overall system description. Thus, FMI has the potential to accelerate the development of multi-physics digital models, which can be utilized in digital twins. Model reduction techniques might be necessary to achieve real-time capability for digital twins or real time orchestrators need to be implemented. Furthermore, standardized interfaces and FMUs offers an easy way of adopting the level of detail of the digital twin, as models can be exchanged without further adjustments. To summarize, a standardized toolbox for Co-Simulation building shows high potential to enable digital twins.

Acknowledgements

The authors would like to thank Aline Luxa and Sebastian Frahm for their contribution of simulation models.

References

Branlard, Emmanuel, Jason Jonkman, and Scott Dana, and Paula Doubrawa (2020) “A Digital Twin Based on OpenFAST Linearizations for Real-Time Load and Fatigue Estimation of Land-Based Turbines.” *J. Phys. Conf. Ser.*, vol. 1618, doi:10.1088/1742-6596/1618/2/022030.

Brickley, Dan, Ramanathan V. Guha, and Brian McBride (2014) “Rdf Schema 1.1. W3C Recommendation.” *World Wide Web Consortium*, vol. 2.

Espinosa-López, Manuel, et al. (2018) “Modelling and Experimental Validation of a 46 KW PEM High Pressure Water Electrolyzer.” *Renewable Energy*, vol. 119, pp. 160–73. doi:10.1016/j.renene.2017.11.081.

Flender International GmbH *Digital Gearbox.* winergy-group.com/en/digital-gearbox. Accessed 6 May 2021.

Fuller, Aidan, Zhong Fan, and Charles Day, and Chris Barlow (2020) “Digital Twin: Enabling Technologies, Challenges and Open Research.” *IEEE Access*, vol. 8, doi:10.1109/ACCESS.2020.2998358.

Glaessgen, Edward, and David Stargel (2012) “The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles.” *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conf.*, Honolulu, Hawaii, 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference. Reston, Virigina2012.

Gomes, Cláudio, et al. (2017) *Co-Simulation: State of the Art.* 1 Feb. 2017, arxiv.org/pdf/1702.00686.

Gruber, Tom. “Ontology.” *Encyclopedia of Database Systems*, edited by Ling Liu and M. Tamer Özsu, Springer New York, 2016pp. 1–3.

Kritzinger, Werner, et al. (2018) “Digital Twin in Manufacturing: A Categorical Literature Review and Classification.” *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1016–22. doi:10.1016/j.ifacol.2018.08.474.

Modelica Association Project FMI *FMI Standard 2.0.* fmi-standard.org/downloads/. Accessed 12 Apr. 2021.

Pimenta, F., et al. (2020) “Development of a Digital Twin of an Onshore Wind Turbine Using Monitoring Data.” *J. Phys. Conf. Ser. (Journal of Physics: Conf. Ser.)*, vol. 1618, doi:10.1088/1742-6596/1618/2/022065.

Schaeffler AG *Digital Services.* www.schaeffler.com/content.schaeffler.com/en/news_media/dates_events/windenergy_hamburg/digital_services/digital_services.jsp. Accessed 6 May 2021.

Tao, Fei, He Zhang, and Ang Liu, and A. Y. C. Nee (2019) “Digital Twin in Industry: State-of-the-Art.” *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2405–15. doi:10.1109/TII.2018.2873186.

Thomas, Philipp, et al. (2014) “The OneWind Modelica Library for Wind Turbine Simulation with Flexible Structure.” *10th International Modelica Conf.*, March 10-12, 2014, The 10th International Modelica Conf. Linköping University Electronic Press, 2014pp. 939–48.

Tygesen, Ulf T., et al. (2018) “The True Digital Twin Concept for Fatigue Re-Assessment of Marine Structures.” *Volume 1: Offshore Technology*, 17.06.2018 - 22.06.2018, Madrid, Spain, ASME 2018 37th International Conference on Ocean, Offshore and Arctic Engineering2018.

van Nguyen, Yvon Besanger, and Quoc Tran, and Tung Nguyen (2017) “On Conceptual Structuration and Coupling Methods of Co-Simulation Frameworks in Cyber-Physical Energy System Validation.” *Energies*, vol. 10, no. 12, doi:10.3390/en10121977.

Object-Oriented Models of Parallel Manipulators

Paolo Campanini[1] Gianni Ferretti[2]

[1]MUSP Lab, Strada Torre della Razza, 29122 Piacenza, Italy, paolo.campanini@musp.it
[2]Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Piazza Leonardo da Vinci 32, 20133 Milano, Italy, gianni.ferretti@polimi.it

Abstract

In this paper, the development of models of parallel manipulator is described, based on components of the Modelica standard library only. At first, the dynamic model of a Delta robot is illustrated and validated with respect to experimental data. Then, the model of a Stewart platform is discussed. Thanks to the symbolic manipulation capabilities of the OpenModelica compiler, the model is then used to automatically generate the inverse dynamics, which is in general is a quite difficult task.

Keywords: Object-oriented modelling; simulation; parallel manipulators; Modelica; DAE systems; closed chains

1 Introduction

Owing to their superior performance compared to serial manipulators in terms of stiffness, positioning accuracy and speed, and load-to-weight ratio, parallel manipulators have recently received growing interest also in industrial applications, for example for machining tasks such as drilling and milling (Escorcia-Hernández et al. 2020), and for PCBs (Printed Circuit Board) assembly (Hesselbach and Kerle 1997).

However, the complex kinematics of parallel manipulators, based on multiple closed loops, on one hand simplifies the inverse kinematics, on the other hand limits the achievable workspace and considerably complicates the calculation of the direct kinematics, and above all the dynamic modeling.

The key concept applied in developing the dynamic model of a parallel manipulator consists in cutting the kinematic loops, modelling the resulting serial tree of subchains and introducing the constraint reactions.

In order to model the serial tree, two approaches can be followed: the Lagrange-Euler (LE) formulation and the Newton-Euler (NE) formulation. The Lagrange-Euler formulation is based on the computation of the kinetic and potential energy of the tree as a whole, while the Newton-Euler formulation computes the dynamics of each link of the tree separately. The NE results naturally in large number of equations and, in this respect, is considered as poorly efficient compared to the LE formulation. The question however is debatable, in fact, the complexity of the computation of the Lagrangian largely increases with the number of bodies involved while, on the other hand, a large number of the equations involved in the NE formulation are actually assignments (Elmqvist and Otter 1994) and it is not a case that the most efficient method to compute the dynamics of serial manipulator is based on the NE formulation (Walker and Orin 1982).

The modelling approaches then essentially differ with respect to the way the kinematic constraints and the reaction forces are taken into account.

The Newton-Euler approach has been applied in (Dasgupta and Mruthyunjaya 1998) and in (Briot and Khalil 2015), where the principle of virtual powers has been considered to remove the constraint forces. The principle of virtual work has been also applied in (Tsai 2000) and, in (Jiao et al. 2019), to a Kane's formulation of motion equations (Kane and Levinson 1983; Yang et al. 2016; Lieh 1994). An efficient formulation of the dynamics of a Stewart parallel manipulator, based on the screw theory, has been recently proposed in (Hou, Zhang, and Zeng 2020), shown to be suitable for application to dynamic model-based control.

The Lagrange-D'Alembert formulation has been applied in (Nakamura and Ghodoussi 1989) and, recently, in (Abo-Shanab 2020), where the Jacobian/Hessian matrices of the constraint equations are derived from the kinetic energy. With this approach the constraint forces (Lagrange multipliers) are removed from the motion equations, by projecting the motion of the system into the directions allowed by the kinematic constraints. Similarly, the Natural Orthogonal Complement (NOC) approach has been proposed in (Angeles and Lee 1988), where the constraint forces are eliminated by multiplying the unconstrained dynamical equations by an orthogonal complement, derived from velocity constraints. The NOC approach has been recently applied to a Newton-Euler formulation of the dynamic model of a parallel Schönflies-motion generator (Karimi Eskandary and Angeles 2018). Lagrangian formulation and virtual work principle have been applied in (Xin, Deng, and Zhong 2016), with the goal to derive an efficient control-oriented dynamic model.

In all the cited approaches, the parallel robot model is developed considering its dynamics as a whole, the derivation process is rather complex (the use of symbolic manipulation tools is often suggested, i.e. MAPLE in (Xin, Deng, and Zhong 2016)), and it is generally difficult to integrate into a multi-domain model, taking into account not only the dynamics of the multibody system but also the dynamics of electromechanical or hydraulic actuation

DOI
10.3384/ecp21181241

devices, elasticity and friction, which have a significant effect on the dynamics of real manipulators. In particular, it has been shown in (Grotjahn, Heimann, and Abdellatif 2004) that friction compensation yields significant improvements on control performance.

The modeling approaches described in the literature are therefore difficult to apply to the creation of Digital Twins (DT), compliant with the Industry 4.0 paradigm. In fact, a DT is something more than a simulation model. Having to be as faithful a replica as possible of the physical device, it must also replicate its structure in the connection of components, often belonging, in whole or in part, to physical domains other than the mechanical domain only. In particular, the development of a DT should take place in the same way as the assembly of the components of the physical system. In this respect, the Lagrangian approach to tree structure modeling is clearly not applicable, as it is based on the calculation of the kinetic and potential energy of the entire mechanical system as a whole. Furthermore, even the manipulations of the system of equations necessary to eliminate the constraint reactions from the equations of motion are in contrast with a true modular approach. On the other hand, object-oriented modeling seems to be particularly suitable for the creation of DT, in which it is able to guarantee a true modular multi-domain approach to modeling (Scaglioni and Ferretti 2018).

This paper describes the development of models of parallel manipulators[1], suitable for the creation of DTs, using an object-oriented approach where:

- Only components of the standard Modelica (Fritzson et al. 2020) library are used (no equations have been written, apart from the explicitly developed inverse kinematics models), connected through the graphical interface of the interpreter (OpenModelica).
- The management of closed kinematic chains is completely transparent to the user and is carried out directly by the symbolic manipulation process during the model compilation phase.
- The constructs of the Modelica language have been used to associate the state variables of the model to the actual degrees of freedom only, i.e. actuators coordinates, and to guide the symbolic manipulation and the calculation of the initial configuration of the simulations.

The description of the mechanical dynamics is therefore distributed over the various components (links), through the Newton-Euler approach, which is often considered in the literature as inefficient, as it implies the explicit calculation of the constraint reactions (useless for control-oriented models) but, on the other hand, this calculation is still important, especially for the use of the DT in the design phase.

At first, the dynamic model of a Delta robot is illustrated and validated with respect to experimental results. Then, the model of a Stewart platform is discussed and used to automatically generate the inverse dynamics, thanks to the symbolic manipulation capabilities of the environment. In other words, an algebraic function calculating the motor torques from the position, velocity and acceleration of motor coordinates is generated, suitable to be used in model-based control strategies. Similar functions have been developed in (Hou, Zhang, and Zeng 2020), just with reference to a Stewart platform, and in (Xin, Deng, and Zhong 2016), where the MATLAB function implementing the inverse dynamics of the 3-DOF parallel manipulator presented in (Huang et al. 2005) has been developed.

The paper is organized as follows. In Section 2 the model of a Delta robot is described and validated with respect to experimental results. In Section 3 the model of a Stewart platform is described. In Section 3.2 the process of generating the inverse dynamics function of the Stewart platform is illustrated. Finally, some Conclusion is given in Section 4.

2 Model of a Delta robot

The Delta robot considered here is the result of project developed in cooperation between *Logicon* and *Mitsubishi Electric Italia*. The servomotors and the relevant motion hardware and software were provided by Mitsubishi Electric Italia, while the mechanical components were purchased by third parties. Logicon was responsible for the assembly of the machine and for all other aspects related to the project, such as the design of the electrical cabinet, wiring, technical drawing and FEM analysis. Figure 1 shows a picture of the robot, while Figure 2 illustrates its kinematics.

The structure of the robot consists of three identical legs connecting a fixed base with a moving platform. Each leg is composed of an upper arm and a parallelogram structure; three actuated revolute joints connect the upper arms with the base and provide motion, while all other joints are spherical joints.

The spherical joints add an additional degree of freedom, corresponding to the rotation of the rods around an axis passing through the centers of the spherical joints. In reality, this degree of freedom is constrained by a system of springs (Figure 3), which however only exerts internal forces and therefore has no influence on the motion. Wanting to describe the rods as rigid bodies, without introducing the modeling of the spring system, it would be necessary to replace two spherical joints with two universal joints. However, since the rods, made of carbon, are thin and light, it is possible to concentrate their mass in the middle of the rod and thus avoid the introduction of the additional degree of freedom.

The top level Modelica model is shown in Figure 4, it includes the world reference (1), the global parameters

[1] The developed models are freely available at `https://github.com/looms-polimi/Parallel_manipulators`.

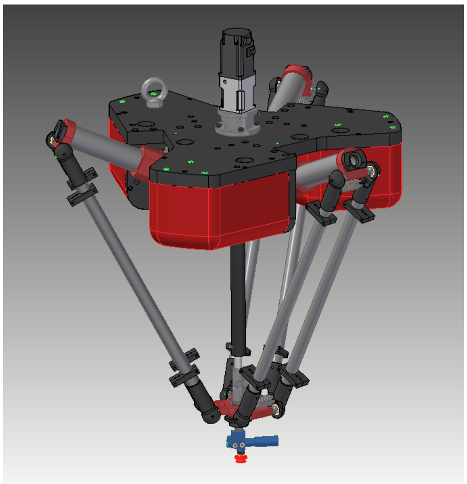


Figure 1. Picture of the Delta robot.

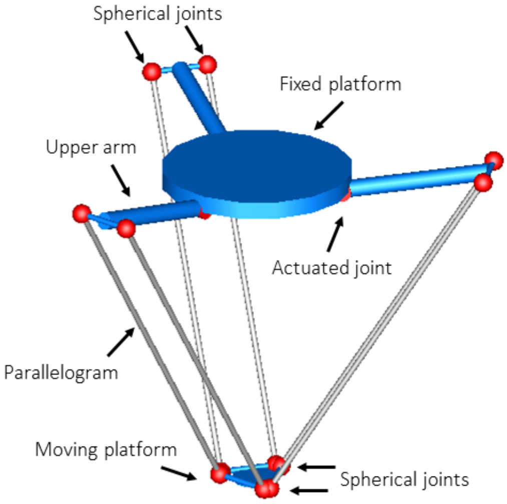


Figure 2. Kinematics of the Delta robot.

record (2), the motion planner (3), the motion controllers (4), the platform (6) and the base (7) models and the aggregate model of the legs (5), shown in Figure 5. The world reference model (1) defines the inertial reference frame and the gravity field (it must be always present when the `package MultiBody` is used (Otter, Elmqvist, and Mattsson 2003), while the global parameters record (2) collects all the main parameters of the model to improve readability and modifiability. The motion planner (3) defines the trajectory of the origin of the platform reference frame; so far linear, trapezoidal and cubic trajectories can be assigned, as well as the pick-and-place trajectory considered for validation. The motion controller model (4) first implements the inverse kinematics, thus computing the reference signals for the controllers of the motor coordinates: 3 identical classical independent PID controllers have been implemented.

It must be noted that controllers are connected to servomotors through an `expandable connector`, denoted by the yellow cable connector. This (input/output) connector models the role of the communication bus on the real machine, collecting the control signals, thus the encoder measurements from the servomotors and the current setpoints from the controllers.

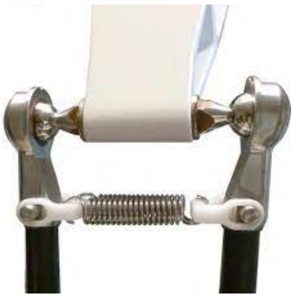


Figure 3. Springs system

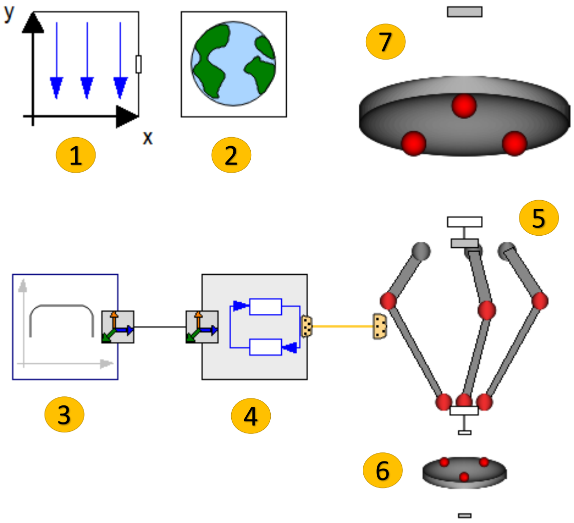


Figure 4. Delta robot model top level

The leg model is defined by the actuator model (Figure 6(a)) connected to the upper arm (rigid body), in turn rigidly connected to the upper short side of the parallelogram (Figure 6(b)). The model of the electrical motor could have been easily included (Ferretti et al. 2002) in the servomotor model, but this would have required the adoption of very short integration step sizes, needed to follow the electrical dynamics of the windings. Since in this work the focus is on the much slower mechanical dynamics, the whole current (torque) control loop has been approximated with a first order transfer function, modelling the torque control loop bandwidth. The gearbox model has been taken directly from the Modelica standard library (Pelchen, Schweiger, and Otter 2002). In particular, the (lossy) gearbox model models the gear ratio and the losses of a standard gear box, including the stuck phases that may occur at zero speed, due to the friction in the gear teeth and/or in the bearings. The loss terms, efficiencies and friction torques, can be arbitrarily defined through lookup tables as functions of the absolute value of the input shaft speed and of the power flow direction. The base

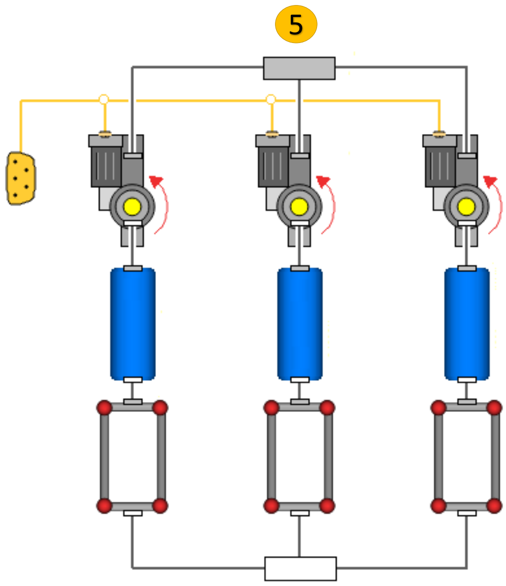


Figure 5. Delta robot legs model

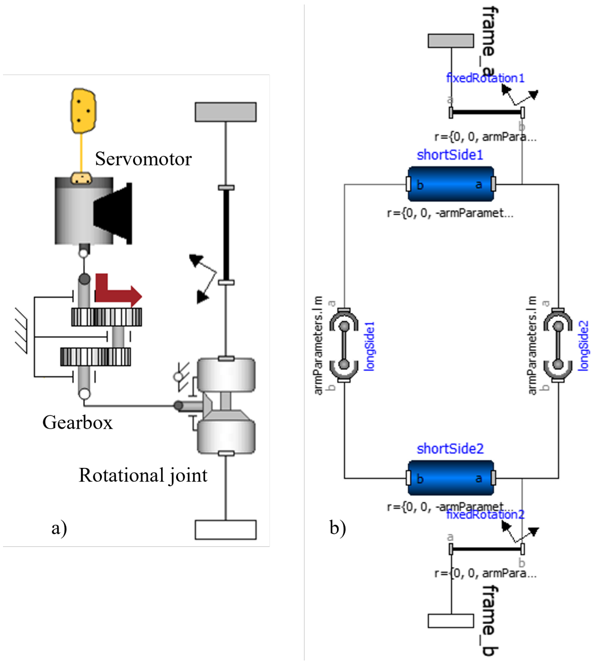


Figure 6. Delta robot leg model

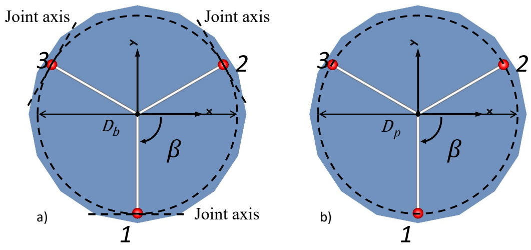


Figure 7. Delta robot base and platform

(6) and the platform (7) can have different shapes and geometries, however, the kinematics is only defined by their attached reference frames, while dynamics is only defined by the inertial parameters of a single rigid body; in this case both base and platform have been modelled as discs (cylinders). The upper arm rotational joints are attached to the base along a circumference of diameter D_b, displaced by 120°, with the rotation axes tangent to the circumference (Figure 7(a)). The platform is attached to the lower sides of the parallelograms through frames placed along a circumference of diameter D_p, still displaced by 120° (Figure 7(b)).

It must be pointed out that the upper and lower connectors of the legs model in Figure 5 are actually vectors of frames. In other words, all the servomotors frames and the frames attached to the lower sides of the parallelograms are collected in a vector, in turn connected to another vector of frames in the base and platform. The base and platform models then define the correct geometrical displacements among the legs connectors through fixed rototranslation models.

The model was built using library joints, which potentially introduce the state variables associated with the introduced degrees of freedom. As a consequence of the closure of the kinematic chains, however, the degrees of freedom are only those associated with the actuators and, properly, the state variables of the model should be associated with the actual degrees of freedom only. To implement this choice and guide the symbolic manipulation, the construct `StateSelect.always` was used for the variables of the actuators (position and speed) and `StateSelect.never` for the variables of all the other joints. Another problem concerns the initialization of the model, in particular the need to solve closed kinematic chains. Also in this case it was decided to set the actuator variables using the `fixed = true` attribute, and then define the initial values of all the other variables as attempt values.

The model has been validated with reference to a fast pick-and-place trajectory, depicted in Figure 8, where the end effector repeatedly travels along a linear trajectory of 82.4 cm in 0.44 s, therefore at an average speed of 1.87 m/s (video available). Since the structure and settings of joint controllers were not available, the validation has been performed by imposing the measured joint velocities to the model (thus computing the inverse dynamics), while comparing the measured joint torques to the simulated ones. Figure 9 shows the measured joint velocities, while Figures 10, 11 and 12 show the comparison between the measured and the simulated torque for joint 1, 2 and 3 respectively.

Although a good correspondence has been obtained between the simulation results and the measurements, some discrepancies are highlighted, in particular in Figure 10 at about time 0.3 s, in the form of a vibratory mode in the experimental data, and in Figures 11 and 12 at about 0.15 s, where the difference between simulations and measurements appears particularly evident.

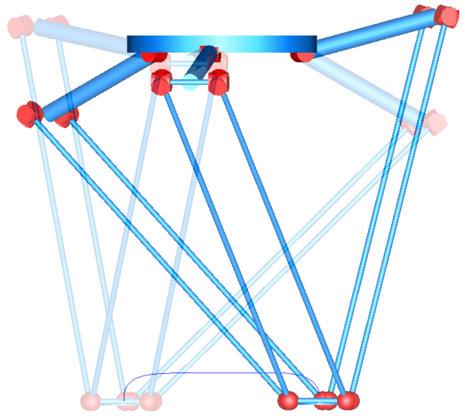


Figure 8. Pick and place trajectory

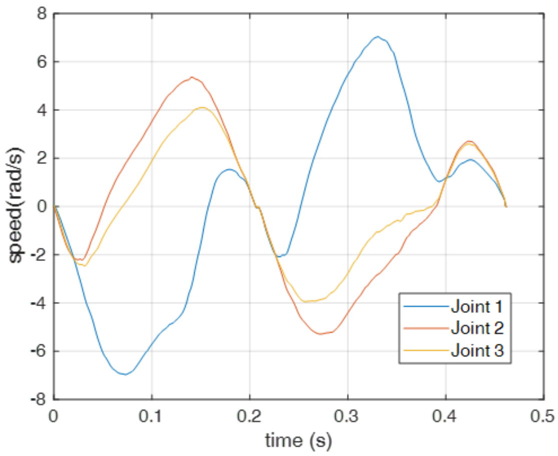


Figure 9. Measured joint velocities

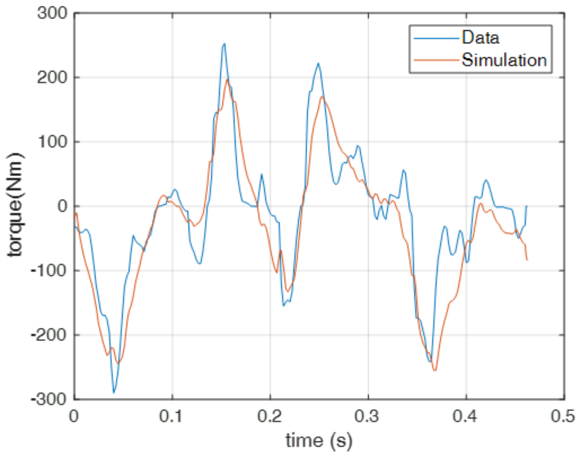


Figure 10. Meas. (blue) and simulated torque (red) on joint 1

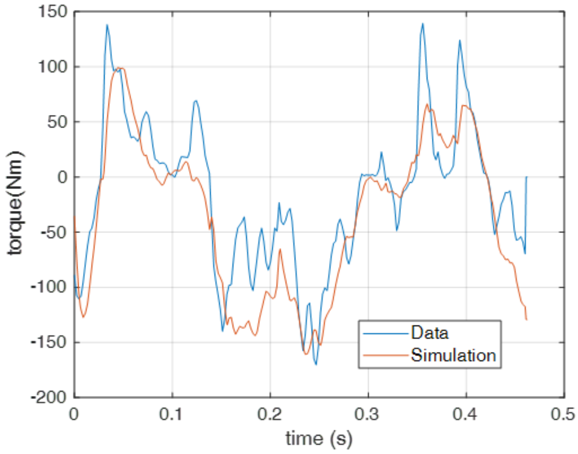


Figure 11. Meas. (blue) and simulated torque (red) on joint 2

The main cause of these discrepancies is probably attributable to the adoption of ideal gear and spherical joint models, not taking into account friction, backlash and elasticity. The vibratory modes detected are most likely due to the compliance of the parallelogram rods, modeled as rigid. It is also possible that at high operating speeds the same structure on which the robot is fixed may introduce vibrations.

3 Model of a Stewart platform

The Stewart platform considered in this work has been designed for an innovative labeller for bottles of different size and dimension, Figure 13(a) shows a picture of the platform, while Figure 13(b) illustrates its kinematics.

It consists of a base and a platform, the former is usually fixed to the ground, while the latter can be positioned in space, in both position and orientation (6 d.o.f.), through 6 identical legs. Each leg is extendable through an electric cylinder, connected to the base through universal joints and to the moving platform through spherical joints. The electric cylinder is a mechanical linear drive unit with a piston rod, the driving component consists of an electrically actuated spindle converting the rotary motion of the motor into a linear motion of the piston rod.

3.1 Direct dynamics

The top level Modelica model is shown in Figure 14(a) and is very similar to the Delta robot top model, while the aggregate model of the legs (5) is shown in Figure 14(b). In this case the motion planner (3) defines the trajectory of platform pose, in turn defined by the position of the origin of the platform reference frame and by the Euler angles relating the orientation of the platform frame with respect to the world frame; so far linear, trapezoidal and cubic trajectories can be assigned. Accordingly, 6 identical classical independent PID controllers have been implemented for the motion of the cylinders. The leg model, whose hierarchy is shown in Figure 15, is defined by the servomotor model connected to the cylinder model. In turn, the cylinder model is defined by the connection of two rigid bodies (stator and rod) through a prismatic joint, connected to the servomotor model through a gearbox model and a screw drive model. The positions of the joints on the base and platform are placed on a circumference, of diameters D_b and D_p, in couples displaced by 120°, with α_b, α_p being

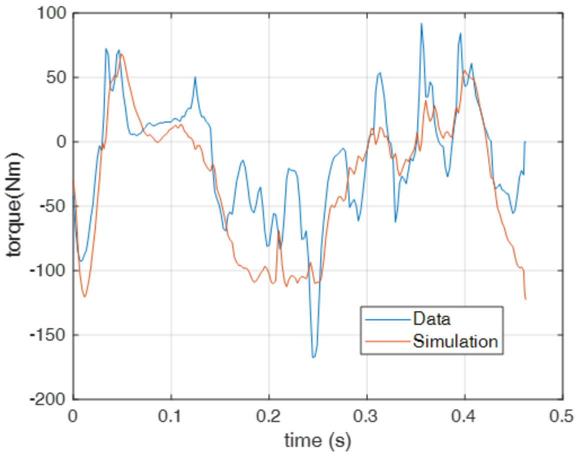


Figure 12. Meas. (blue) and simulated torque (red) on joint 3

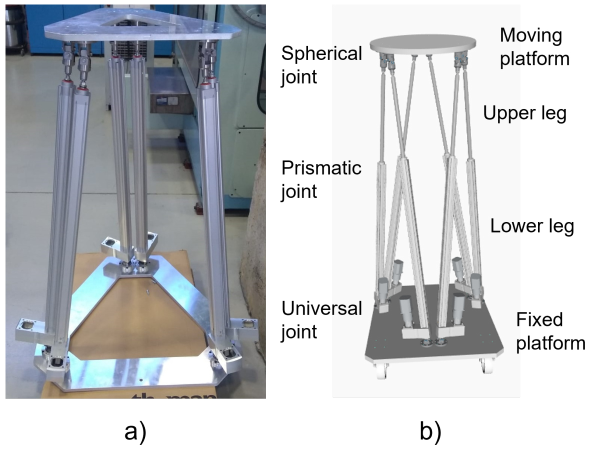


Figure 13. Stewart platform

the angular displacements between the positions of each couple (Fig. 16). The right and left connectors of the legs model in Fig. 14(b) are again vectors of frames, connected to other vectors of frames on the base and platform.

3.2 Inverse dynamics

The use of the model of the Stewart platform has been investigated for the automatic generation of the inverse dynamics, namely of an algebraic function computing the torques as a function of joints positions, velocities and accelerations, mainly used in control and trajectory planning (Balafoutis and Patel 1991). In control applications, inverse dynamics is usually applied to convert positions, velocities and accelerations, computed according to some desired trajectory, into the joint generalized forces which will achieve the desired motion. In trajectory planning, inverse dynamics can be used to check that the desired trajectory can be executed without exceeding the actuators' limits. Analytically computing the inverse dynamics is a difficult task, particularly in the case of closed kinematic chains but, thanks to the symbolic manipulation capabilities of the Modelica interpreter, the inverse dynamics can be automatically generated from the model.

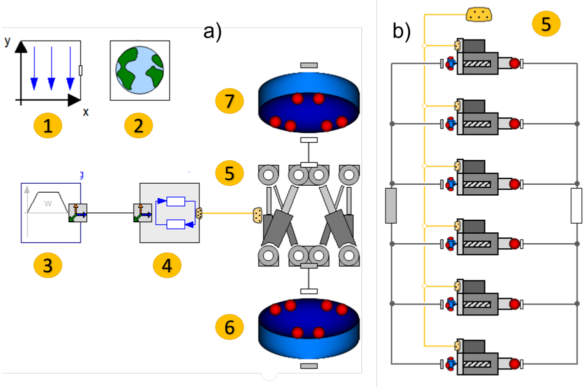


Figure 14. Stewart platform

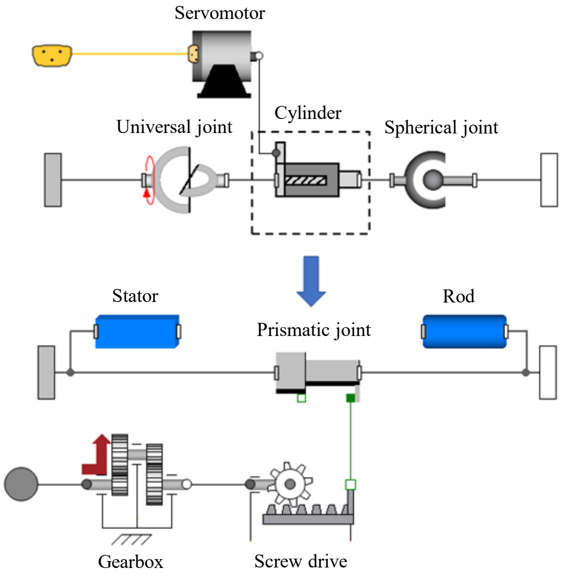


Figure 15. Stewart platform: leg model

The torques computed by the inverse dynamics have been checked along the trajectory depicted in Figure 17, obtained by controlling joint positions through 6 independent PID controllers. The model of the Stewart platform has been then copied and pasted (Figure 18 and 19), while modifying the input structure. Thanks to the `AngleToTorqueAdaptor` model (Figure 20), instead of the joint torques, the inputs are defined by the joint positions, velocities and accelerations computed by the direct dynamics. The differences δ_i ($i = 1,\ldots,6$) between the torques computed by the direct dynamics and the torques computed by the the inverse dynamics is shown in Figure 21, as it can be seen it is equal to zero (numerically $\pm 1.5 \times 10^{-16}$ N). A standalone model can be also generated (Figure 22).

It is important to point out that the inverse dynamic model was obtained directly, without any manipulation, from the direct dynamic model, which in turn was built in a modular way using only library models. Instead, all the approaches mentioned in the introduction required a

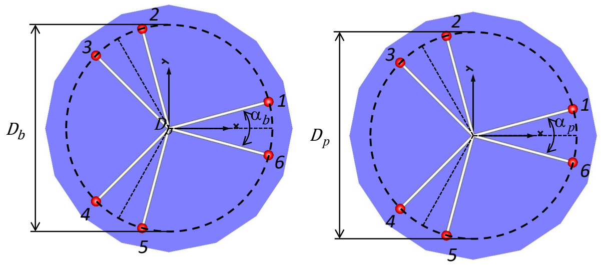


Figure 16. Stewart platform base and kinematic parameters

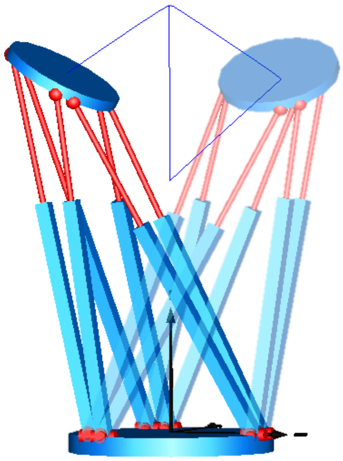


Figure 17. Stewart platform trajectory

specific analytical approach, and in some cases the use of symbolic manipulators. However, if on the one hand the efficiency in the construction of the inverse dynamic model can be considered as optimal, on the other hand the question of the computational efficiency of the generated model remains open, to be investigated in future works.

4 Conclusion

In this paper, the development of models of parallel manipulators based on an object-oriented modelling approach is discussed. The models have been developed based on components of the Modelica standard library only, without writing a single line of code. Two different manipulator have been considered: a Delta robot and a Stewart platform. The model of the Delta robot has been validated on the basis of experimental data, collected on a real robot. The model of the Stewart platform has been considered in order to automatically generate the inverse dynamics model, which is in general a quite difficult task to be performed manually. The main future development seems to be the verification of the numerical efficiency of the automatically generated inverse dynamics model, for the purpose of real time control and planning.

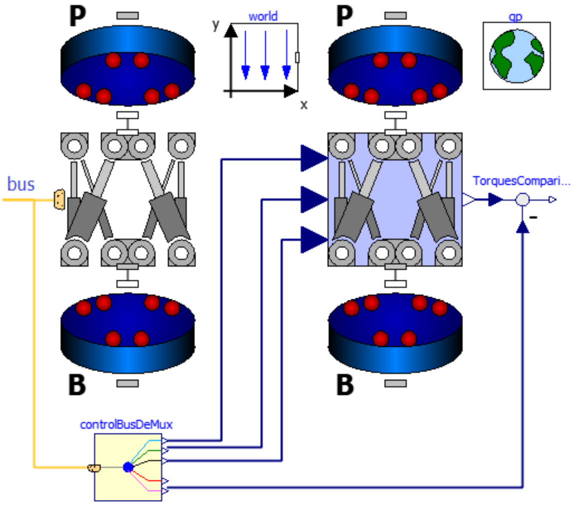


Figure 18. Stewart platform: direct and inverse dynamics

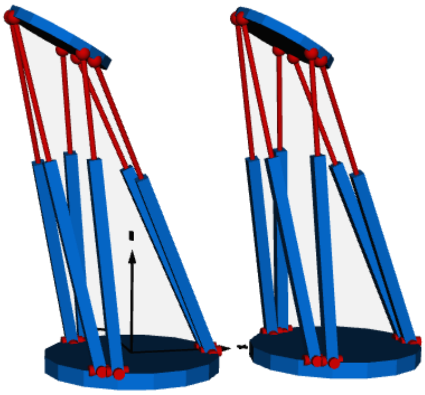


Figure 19. Stewart platforms

References

Abo-Shanab, R. F. (2020). "Dynamic modeling of parallel manipulators based on Lagrange–D'Alembert formulation and Jacobian/Hessian matrices". In: *Multibody System Dynamics* 48.4, pp. 403–426.

Angeles, J. and S. K. Lee (1988). "The formulation of dynamical equations of holonomic mechanical systems using a natural orthogonal complement". In: *Journal of Applied Mechanics* 55.1, pp. 243–244.

Balafoutis, C. A. and R. V. Patel (1991). "Manipulator Inverse Dynamics". In: *Dynamic Analysis of Robot Manipulators: A Cartesian Tensor Approach*. Boston, MA: Springer US, pp. 117–182.

Briot, S. and W. Khalil (2015). *Dynamics of parallel robots - From rigid bodies to flexible elements*. Springer.

Dasgupta, B. and T. S. Mruthyunjaya (1998). "Closed-form dynamic equations of the general Stewart platform through the Newton–Euler approach". In: *Mechanism and Machine Theory* 33.7, pp. 993–1012.

Elmqvist, H. and M. Otter (1994). "Methods for tearing systems of equations in object oriented modeling". In: *Proceedings of the 1994 European Simulation Multiconference, ESM94*. Barcelona, Spain, pp. 1–7.

Escorcia-Hernández, J. M. et al. (2020). "A new solution for machining with RA-PKMs: Modelling, control and experiments". In: *Mechanism and Machine Theory* 150, pp. 1–18.

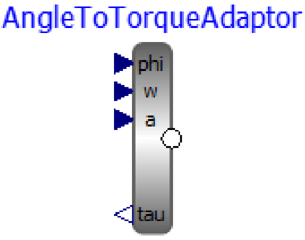


Figure 20. `AngleToTorqueAdaptor` model

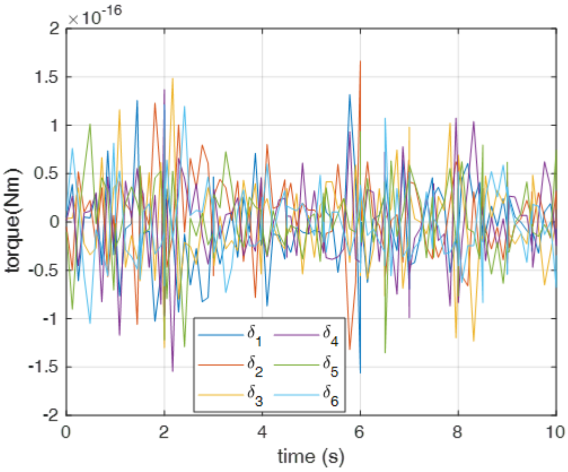


Figure 21. Difference between direct and inverse torques

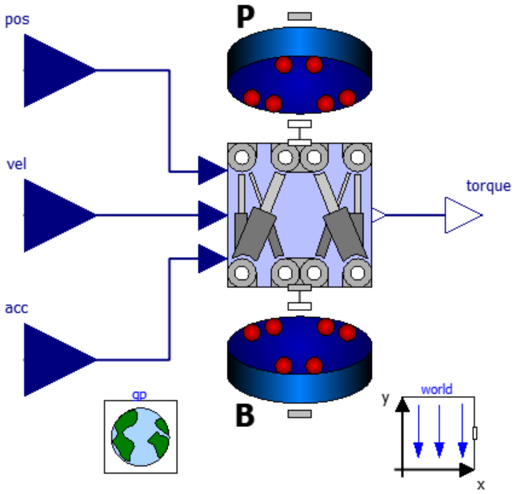


Figure 22. Inverse dynamics model

Ferretti, G. et al. (2002). “Simulating permanent magnet brushless motors in DYMOLA”. In: *Proceedings of the 2nd International Modelica Conference*. Oberpfaffenhofen, Germany, pp. 109–115.

Fritzson, Peter et al. (2020). “The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development”. In: *Modeling, Identification and Control* 41.4, pp. 241–295.

Grotjahn, M., B. Heimann, and H. Abdellatif (2004). “Identification of friction and rigid-body dynamics of parallel kinematic structures for model-based control”. In: *Multibody System Dynamics* 11.3, pp. 273–294.

Hesselbach, J. and H. Kerle (1997). “Placing SMD parts on electronic circuit boards with parallel robots - Concepts and calculatory design”. In: *Proceedings of IFAC Workshop on Intelligent Manufacturing Systems, IMS'97*. Vol. 30. 14. Seoul, Korea, pp. 223–228.

Hou, Y., G. Zhang, and D. Zeng (2020). “An efficient method for the dynamic modeling and analysis of Stewart parallel manipulator based on the screw theory”. In: *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* 234.3, pp. 808–821.

Huang, T. et al. (2005). “Conceptual design and dimensional synthesis for a 3-DOF module of the TriVariant-a novel 5-DOF reconfigurable hybrid robot”. In: *IEEE Transactions on Robotics* 21.3, pp. 449–456.

Jiao, Jian et al. (2019). “Dynamic modeling and experimental analyses of Stewart platform with flexible hinges”. In: *Journal of Vibration and Control* 25.1, pp. 151–171.

Kane, T. R. and D A. Levinson (1983). “The use of Kane's dynamical equations in robotics”. In: *The International Journal of Robotics Research* 2.3, pp. 3–21.

Karimi Eskandary, P. and J. Angeles (2018). “The dynamics of a parallel Schönflies-motion generator”. In: *Mechanism and Machine Theory* 119, pp. 119–129.

Lieh, J. (1994). “Computer-oriented closed-form algorithm for constrained multibody dynamics for robotics applications”. In: *Mechanism and Machine Theory* 29.3, pp. 357–371.

Nakamura, Y. and M. Ghodoussi (1989). “Dynamics computation of closed-link robot mechanisms with nonredundant and redundant actuators”. In: *IEEE Transactions on Robotics and Automation* 5.3, pp. 294–302.

Otter, M., H. Elmqvist, and S. E. Mattsson (2003). “The new Modelica multiBody library”. In: *Proceedings of the 3rd International Modelica Conference*. Linköping, Sweden, pp. 311–330.

Pelchen, C., C. Schweiger, and M. Otter (2002). “Modeling and simulating the efficiency of gearboxes and of planetary gearboxes”. In: *Proceedings of the 2nd International Modelica Conference*. Oberpfaffenhofen, Germany, pp. 257–266.

Scaglioni, B. and G. Ferretti (2018). “Towards Digital Twins through object-oriented modelling: a machine tool case study”. In: *Proceedings of the 9th Vienna International Conference on Mathematical Modelling, MATHMOD2018*, pp. 613–618.

Tsai, L. (2000). “Solving the inverse dynamics of a Stewart-Gough manipulator by the principle of virtual work”. In: *Journal of Mechanical Design* 122.1, pp. 3–9.

Walker, M. W. and D. E. Orin (1982). “Efficient dynamic computer simulation of robotic mechanisms”. In: *Journal of Dynamic Systems, Measurement, and Control* 104.3, pp. 205–211.

Xin, G., H. Deng, and G. Zhong (2016). “Closed-form dynamics of a 3-DOF spatial parallel manipulator by combining the Lagrangian formulation with the virtual work principle”. In: *Nonlinear Dynamics* 86, pp. 1329–1347.

Yang, J. et al. (2016). “Dynamic modeling and control of a 6-DOF micro-vibration simulator”. In: *Mechanism and Machine Theory* 104, pp. 350–369.

A Modelica Library for Modelling of Electrified Powertrain Digital Twins

Nikolaos Fotias[1] Ran Bao[2] Hui Niu[2] Michael Tiller[3]
Paul McGahan[1] Adam Ingleby[2]

[1]Ricardo Prague s.r.o., Czech Republic, {Nikolaos.Fotias, Paul.McGahan}@ricardo.com

[2]Ricardo UK Ltd., UK, {Ran.Bao, Hui.Niu, Adam.Ingleby}@ricardo.com

[3]Ricardo Inc., USA, Michael.Tiller@ricardo.com

Abstract

In this paper, a Modelica library of electrified powertrain components is presented and its applications discussed. This library is used to construct digital twins of electrified powertrains during product development. These digital twins provide value by reducing development time and cost, while once the product is in-service, they enable improved condition monitoring. The library includes a multi-fidelity and multi-scale battery and power electronics sub-library, an Electrical Drive Unit (EDU) sub-library modelling different types of electrical machines, and an electrified propulsion system sub-library of template models that leverage the battery, power electronics and EDU components found in the other sub-libraries. Finally, an example of applying the proposed library to electrified vehicle development is presented.

Keywords: Modelica, Digital Twins, Electrified Powertrain

1 Introduction

Digitalisation is revolutionising the complete product lifecycle: from development and production to testing, in-service maintenance and recycling. “Digital Twin” (DT) technology will bring a significant reduction in electric powertrain development time, cost and risk: through up-front design analysis, optimisation and testing in a virtual environment, without the need for multiple prototypes. At its core, a DT is a representation of a physical product that can be used as a testing ground for monitoring, simulating and optimizing design and operational performance.

This paper describes the work done in a project which had the key objective of assessing the impact of DT techniques on product development. In this project, the first focus was developing Digital Twins for each sub-system (Battery, Power Electronics and EDU) in the electrified powertrain. The second focus was the integration of each sub-system, to create an Electrified Powertrain Digital Twin. Such a system level Digital Twin can be used, as part of the virtual product development process, in the design and optimisation of the electrified powertrain.

We have created an Electrified Powertrain Modelica library, called ePropulsionSystem, which incorporates necessary plant models to enable development of Digital Twins. The novelty of this developed library arises from the ability to seamlessly swap between model variants and fidelity levels, while offering an interface to couple with already existing libraries to streamline concept development, with the ultimate goal of effectively linking the developed virtual model to the physical model.

In this paper we will discuss the various models included in the different sub-libraries as well as an example of applying the `ePropulsionSystem` model to an Electric Vehicle (EV) use case.

1.1 Literature Review

The battery models describe not only the electrical behaviour of the battery but also the thermal response and aging characteristics (Einhorn, et al., 2011; Gerl, Janczyk, Krüger, & Modrow, 2014; Surewaard, Karden, & Tiller, 2003; Dao & Schmitke, 2015; Bao, Fotias, & McGahan, 2021). The most common representations of battery electrical behaviour are electrochemical models and Equivalent Circuit Models (ECM) (Fan, Pan, Bartlett, Canova, & Rizzoni, 2014; Perez, Shahmohammadhamedani, & Moura, 2015; Guo, Jin, & White, 2017; He, Xiong, & Fan, 2011; Chen & Rincon-Mora, 2006). It is well known that the overall battery pack performance is influenced significantly by the battery temperature (McGahan, Rouaud, & Booker, 2019) which we model using a network of 1D lumped thermal components (Pesaran, 2002; Johnson, Pesaran, & Sack, 2000; Park & Jaura, 2003; Nelson, Dees, Amine, & Henriksen, 2002). Investigation of battery aging mechanisms is currently a hot topic in both academia and industry. Calendric ageing and cyclic ageing are two commonly used ageing model. The cause and effect of various battery ageing mechanisms is discussed in detail in (Vetter, et al., 2005). Estimating the battery parameters from measured data is also an important feature in several battery libraries (Gerl, Janczyk, Krüger, & Modrow, 2014; Dao & Schmitke, 2015; Qin, Li, Wang, & Zhang, 2019).

To enable easy model parametrisation and development, while still ensuring high fidelity, behavioural modelling was used to develop the components of the power electronics library. The work was based on concepts presented in the literature (Denz, Schmitt, & Andres, 2014; Cellier, Clauß, & Urquía, 2007; Lai, Hill, & Suchato, 2019; Urkizu, et al., 2019), where the response of the system is an amalgamation of the static response and an estimation of the losses, either using analytical solutions, measurement data or the dynamic model of the system. Given the dependence of the device response on the temperature, the developed models were enhanced to capture that behaviour and thermal models for the devices were developed in accordance with (AG, 2020).

The five most common electric machine technologies among Hybrid Electric Vehicles (HEVs) and EVs are Induction Machines (IMs), Switched Reluctance Motors (SRMs), wound-rotor Synchronous Machines (SMs), Direct Current (DC) machines, and Synchronous Permanent Magnet Machines (SMPM) (Bazzi, 2013; Dorrell, Knight, Popescu, Evans, & Staton, 2010) each with their own advantages and disadvantages. In late 2020, Tesla pioneered a hybrid motor type in the Tesla Model 3, combining characteristics of Interior Permanent Magnet (IPM) motors and Synchronous Reluctance Motors (SynRM) to form the hybrid IPM-SynRM motor.

While an IPM machine demonstrates high efficiency at high speed, this comes at the cost of output torque (Hwang, Han, Kim, & Cha, 2018). By comparison, IPM-SynRM devices possess better efficiency at low speed and better thermal efficiency than traditional SynRMs subject to temperature limits (Ramakrishnan, Stipetic, Gobbi, & Mastinu, 2018; Xing, Sun, & Lei, 2014; Lee, Kim, Jung, Hong, & Kim, 2012; Haumer & Kral, Motor Management of Permagnent Synchronous Machines, 2012). Since the IPM is a type of SMPM, this project focused on the development of SMPM and SynRM (abbreviated as SMR in Modelica nomenclature) motors.

There are several software packages available for the mathematical modelling and simulation of EVs and EDUs, including MATLAB/Simulink, Simpower, Python based models and Modelica (Mohd, Hassan, & Aziz, 2015; McDonald, 2012). Because the Modelica Standard Library (MSL) is open source and provides a library of multi-domain physical models found in automotive components, the MSL can be an excellent starting point for the development of EDU digital twins (Einhorn, et al., 2111). The MSL has a basic electric machine library which includes the traditional asynchronous Induction Machines and Synchronous Machines (Ceraolo, 2015). The MSL and other Modelica libraries also have extended models that consider friction losses and include thermal effects, *e.g.*, the Fundamental Wave library (Kral & Haumer, 2011), Advanced library (Haumer, Kral, Kapeller, Bäuml, & Gragger, 2009) and SmartDrive library (Gragger, Kral, Hansjörg, & Pirker, 2006).

2 Library Structure

Figure 1 shows the typical architecture of the digital twin models created using our `ePropulsionSystem` library. As shown in Figure 1, the Battery, Power Electronics and EDU are the key components being modeled. This library does not include models of the controllers for these devices as we have chosen to focus exclusively on plant models in this library. The `ePropulsionSystem` library is structured as shown in Figure 2.

2.1 Battery

The key models in the `Battery` sub-library are the cell level `Electrical` and `Ageing` models. These cell level models are then leveraged in the `BatteryPack` models. These models are fully developed and will be discussed in detail later in this paper. The `Battery` sub-library also includes `Thermal`, `Cooling`, `Ancillaries` and `BatteryManagementSystem` component libraries which will be developed in the future and will not be discussed in this paper.

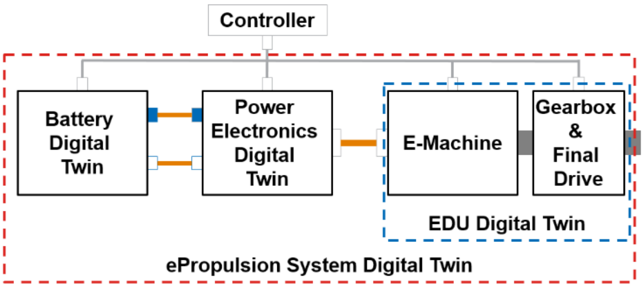


Figure 1. Architecture of a typical `ePropulsionSystem` digital twin model

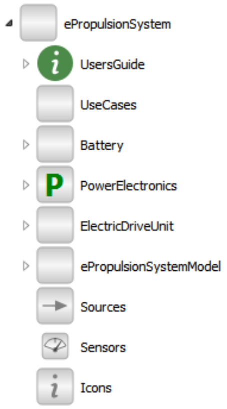
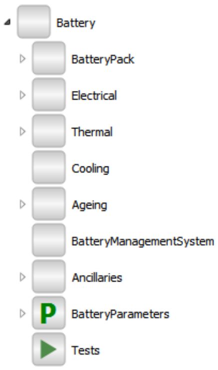


Figure 2. `ePropulsionSystem` library structure



- Battery
  - BatteryPack
  - Electrical
  - Thermal
  - Cooling
  - Ageing
  - BatteryManagementSystem
  - Ancillaries
  - BatteryParameters
  - Tests

Figure 3. The structure of the `Battery` sub-library in `ePropulsionSystem` library

2.1.1 `Electrical`

At present, there are three different electrical cell models included in the `Battery` sub-library. These are the Ideal Voltage Source Model, Internal Resistance Model, and RC Element Model. Schematics for each of these models are shown in Figure 4. Each of these electrical cell models can be characterized by how their Open Circuit Voltage (OCV) and internal resistance are modeled. For this reason, a template model for electrical cells was created and is shown in Figure 5.

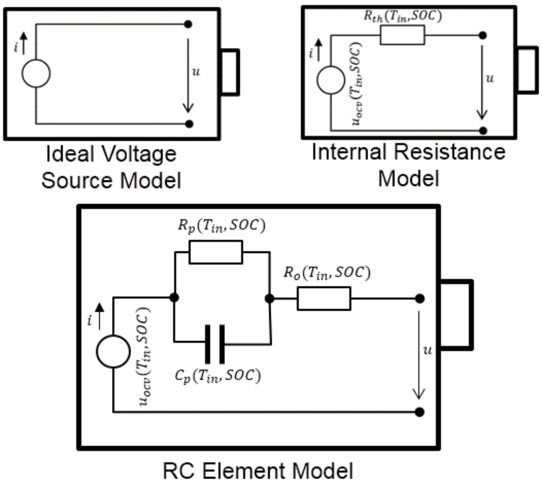


Figure 4. Different electrical cell models included in the `Battery` sub-library

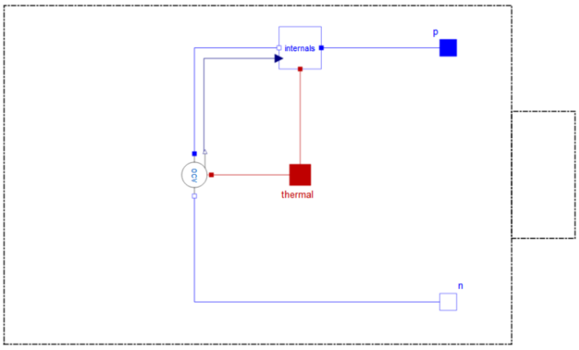


Figure 5. `ElectricalCellTemplate` model

The Electrical Cell Model Template includes a positive connector (`p`), a negative connector (`n`), a thermal connector (`thermal`) as well as the OCV and internal resistance component models. To use one template to represent three different cell electrical models, the OCV and internal resistance component models are made `replaceable`. So, a Modelica package, called `ElectricalInterfaces`, was created which includes the interfaces for the OCV and internal resistance components. Different OCV and internal resistance models, which extend from the `partial` models in the `ElectricalInterfaces` package, were built and included in a package called `ElectricalComponents`. The structure of an electrical cell model in the `Battery` package is shown in Figure 6.

In `ElectricalComponents`, two different OCV models were developed. The first model, `IdealOCV`, uses a constant voltage source for the open circuit voltage. The other model, `TzTableOCV`, computes the OCV based on the cell temperature and State-of-Charge (SOC). The cell SOC is also calculated in the `TzTableOCV` model by integrating the current flow in and out of the cell.

Three different models of the internal resistance of the cell were developed. These are the `IdealShort` model, the `TzTableR`, and the `TzTableRCR` model. All of these models are found in the `Battery.ElectricalComponents` sub-library. The `IdealShort` model assumes there is no internal resistance in the cell. The `TzTableR` and `TzTableRCR` models compute the internal resistance of the cell based on cell temperature and SOC. The topology of the `TzTableR` features only a single (temperature and SOC dependent) resistance while the topology of the `TzTableRCR` includes two resistors and a capacitor (all of which depend on temperature and SOC).

As shown in Figure 4, by starting with the `ElectricalCellTemplate` model and picking different OCV and internal resistance sub-models, a number of different electrical cell models can be created.

In this way, using the `ElectricalCellTemplate` model, the three pre-defined electrical cell models, shown in Figure 6, were created: `IdealCell`, `R_Cell` and `RC_Cell`.

2.1.2 `Ageing`

Normally, battery ageing models includes two modes of ageing. These are cycling and calendaring. In this library, only the cycling ageing model is considered. A cycling ageing model is a semi-empirical model of two main effects of ageing on cell performance, capacity fade and power fade.

It has been shown experimentally that the capacity fade can be described using a power law with energy throughput and is related to temperature via an Arrhenius relationship. This means that the capacity loss (capacity fade), can be described using the equation below (Andrea Cordoba-Arenas, 2015):

$$Q_{fade} = k_{Q_{sev}}\, Ah^{z} \qquad (1)$$

Where, Q_{fade} is the capacity fade [%]; $k_{Q_{sev}}$ is the capacity fade severity factor [-]; Ah is the charge throughput [kAh]; z is the power exponent [-].

The resulting increase in internal resistance can be described using a very similar equation to capacity fade, however without the power exponent (Andrea Cordoba-Arenas, 2015):

$$IR_{inc} = K_{IR_{sev}}\, Ah \qquad (2)$$

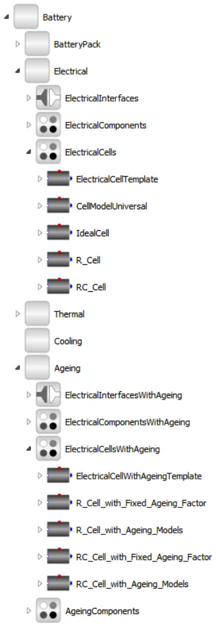
Where, IR_{inc} is the increase in internal resistance [%]; $K_{IR_{sev}}$ is the increase in internal resistance severity factor [-]; Ah is the charge throughput [kAh].

The Ageing model was created by using Equation (1) and (2) and added to the `ElectricalCellTemplate` model to create a new template called `ElectricalCellsWithAgeing` as shown in Figure 7. As the Ageing model reduces the cell capacity, and increases the cell internal resistance, the `ElectricalInterfaces` package was updated by adding new inputs and outputs to create a new package called `ElectricalInterfacesWithAgeing` in the `Ageing` package; and the `ElectricalComponentsWithAgeing` was updated to include the calculation of ageing effects from the `ElectricalComponents` package. The structure of the `Ageing` package is presented in Figure 6. Again, we can choose models for the OCV and internal losses to create several different electrical cell models some of which are shown in Figure 6.

2.1.3 BatteryPack

Using the electrical cell model interface, we can then construct a battery `PackModel` which is composed of battery cells connected both in series and in parallel. This battery `PackModel` and its parameters are shown in Figure 8.

In the `PackModel`, the number of battery cells connected in series and the number of battery cells in parallel are defined by two parameters, `ns` and `np`, respectively Using these parameters in conjunction with the array and looping capabilities in Modelica we are then able to automatically generate the serial and parallel connections needed to wire together every cell in the `PackModel`. Different electrical cell models (with or without ageing) from the `Electrical` and `Ageing` package can be chosen as the `replaceable CellModelType` model which is used to instantiate each cell in the `PackModel`. Through our template models we can create both topological and parametric variants to match the underlying cells as shown in Figure 3. We place these fully parameterized models in the `Battery.Parameters` library. In this way, we can create pack models using any of the parameterized cell models that we have developed (with or without ageing) and independently specify the pack topology as well.



- Battery
  - BatteryPack
  - Electrical
    - ElectricalInterfaces
    - ElectricalComponents
    - ElectricalCells
      - ElectricalCellTemplate
      - CellModelUniversal
      - IdealCell
      - R_Cell
      - RC_Cell
  - Thermal
  - Cooling
  - Ageing
    - ElectricalInterfacesWithAgeing
    - ElectricalComponentsWithAgeing
    - ElectricalCellsWithAgeing
      - ElectricalCellWithAgeingTemplate
      - R_Cell_with_Fixed_Ageing_Factor
      - R_Cell_with_Ageing_Models
      - RC_Cell_with_Fixed_Ageing_Factor
      - RC_Cell_with_Ageing_Models
    - AgeingComponents

Figure 6. The structure of `Electrical` and `Ageing` packages

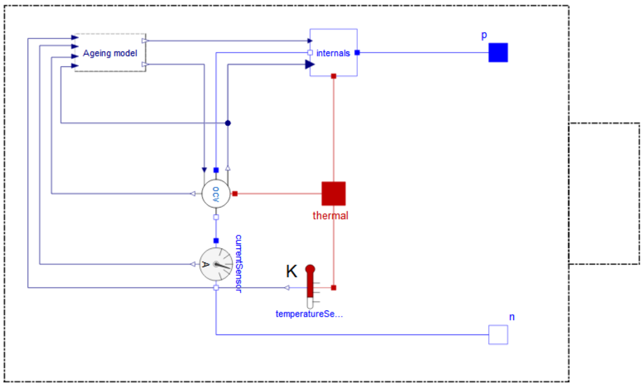
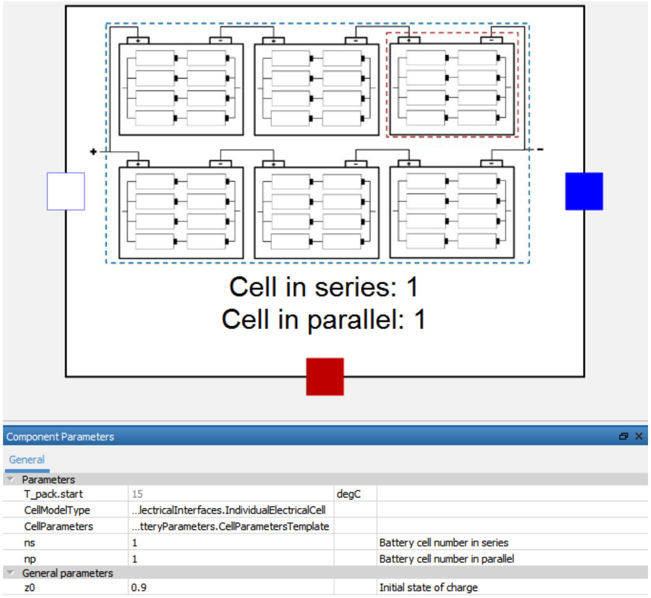


Figure 7. `ElectricalCellsWithAgeing` template



Component Parameters

General

Parameters			
T_pack.start	15	degC	
CellModelType	...lectricalInterfaces.IndividualElectricalCell		
CellParameters	...tteryParameters.CellParametersTemplate		
ns	1		Battery cell number in series
np	1		Battery cell number in parallel
General parameters			
z0	0.9		Initial state of charge

Figure 8. Battery `PackModel` and its parameters

2.2 `PowerElectronics`

The `PowerElectronics` sub-library is a part of the overall Digital Twin for Electrified Powertrains library, `ePropulsionSystem`, and focuses on capturing the behavior of the most commonly used electronic components and topologies, while ensuring easy parametrization from available manufacturer data.

The main focus of the library is the modelling of the behavior of Diodes, Insulated Gate Bipolar Transistors (IGBTs) and Metal-Oxide-Semiconductor Field Effect Transistors (MOSFETs) in a way that enables the user to simulate a variety of electronics topologies utilizing any available parametrization data. For that purpose, the developed models are made up of numerous variants to estimate the static response (static model variants) as well as the corresponding losses of the device in question (loss model variants).

The static model variants for the different semiconductors capture the response of the device disregarding the dynamic behavior due to the presence of capacitances and inductances. The models were developed and organized according to the level of detail that is needed for their parametrization:

- Ideal models, where the devices under investigation are represented by their ideal equivalent circuit.
- Constant models, where parameters such as channel resistance are not affected by system variables.
- Look-up table-based models, where the device parameters at each simulation step are the output of a system variable dependent look-up table. Such variables include device temperature, gate voltages etc.

All these variants are developed using the same underlying interface and can be seamlessly swapped using `replaceable` models.

The loss model variants estimate the conduction losses as well as the switching losses of the devices, given a selected static model and the data available to the user. The variants developed are the following:

- Lossless models, which are implemented for fast simulations where heat dissipation is of no interest.
- Constant models, where parameters such as energy release during a switching event are not affected by system variables.
- Look-up table-based models, where parameters, such as the switching energy loss, are estimated at every simulation step from the device variables using look-up tables. Such variables are the device current, the blocking voltage etc.
- Dynamic models, where the dynamic behavior, due to parasitic capacitances and inductances of the device, is captured. The switching and conduction losses are subsequently estimated from the dynamic response of the system.

The same development approach was used as with the static models when integrating these variants together to ensure simple and fast swapping between model variants.

Table 1 summarizes the static and loss model variants as well as the number of possible model representations given those variants

The models described are used as the building blocks of more complex power electronics topologies. An example of the utilization of the models is presented in Figure 9.

The topology under investigation is an isolated DC-DC converter that represents the on-board charger of an EV. Using the models developed we were able to simulate the response of the converter when coupled with a simple PI controller that regulates the output current. Furthermore, we were also able to carry out an investigation on the impact of different semiconductor technologies on the performance of the charger. The output current and voltage curves, as well as the full-bridge semiconductor temperatures are provided in Figure 10, Figure 11, Figure 12 and Figure 13.

Table 1. `PowerElectronics` library model variants

	Static Variants	*Loss Variants*	*Total Variants*
Diodes	4	2	8
IGBTs	5	9	45
MOSFETs	5	9	45

Moreover, we were able to use the developed system to execute batch simulations to estimate the impact of different switching frequencies on the Full-Bridge losses as presented in Figure 14. This capability to automatically execute batch simulations can be used in conjunction with Design of Experiments techniques to carry out system level parameter optimizations.

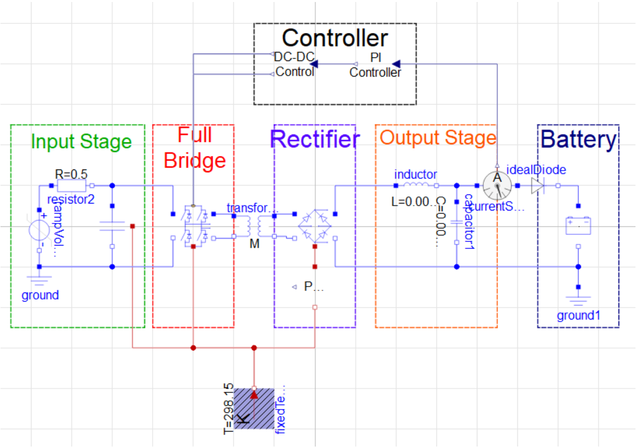


Figure 9. Hard-switched isolated DC-DC converter

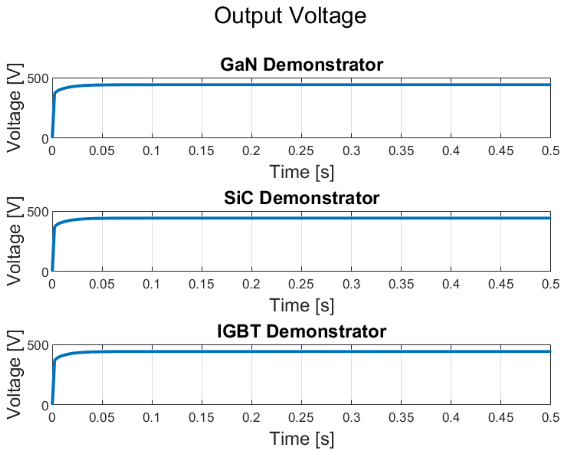


Figure 10. DC-DC Output voltage

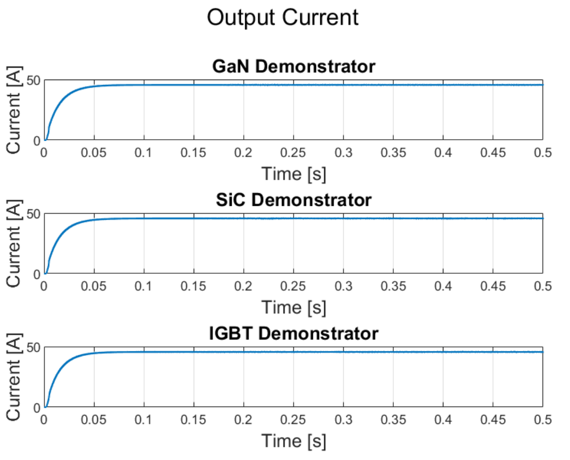


Figure 11. DC-DC Output current

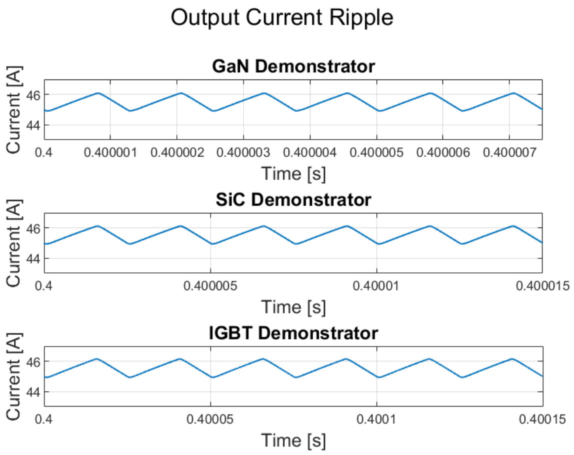


Figure 12. DC-DC Output current ripple

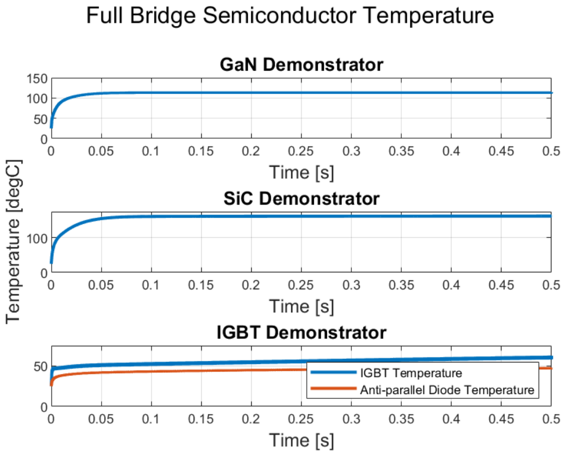


Figure 13. Full bridge semiconductor temperature

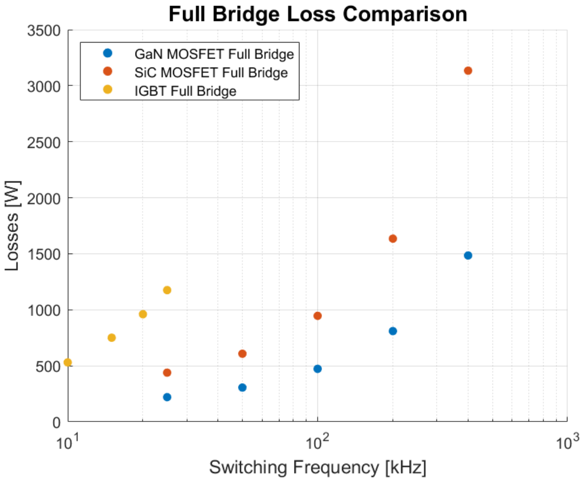


Figure 14. Full bridge technologies loss comparison

2.3 EDU

The electrical machines library found in the MSL (version 3.2.1) contains models for synchronous induction machines, including permanent magnet and synchronous reluctance motors. To develop the electrified powertrain digital twin, the Modelica models were used as templates and extended to form the `EDU` Modelica Library.

2.3.1 Python Automated Data Importing

The library is coupled with the Ricardo eMotor design database (eMAD, see Figure 15) using a Python script. The script automates data transfer and formatting and the process can be broken down as follows: 1) The Python script queries the motor requirement such as motor maximum power, DC voltage, etc. from the eMAD database 2) The user selects a specific MotorCAD design which meets the requirement. 3) The script extracts from eMAD detailed parameters such as d-q inductances, pole numbers, the open circuit voltage, nominal frequency etc. (see Table 2) and saves the result into a Modelica parameter file. 4) The user directly imports the file into Modelica using a `function` called `Modelica.Utilities.Example.readRealParameter` from MSL.

2.3.2 EDU library structure

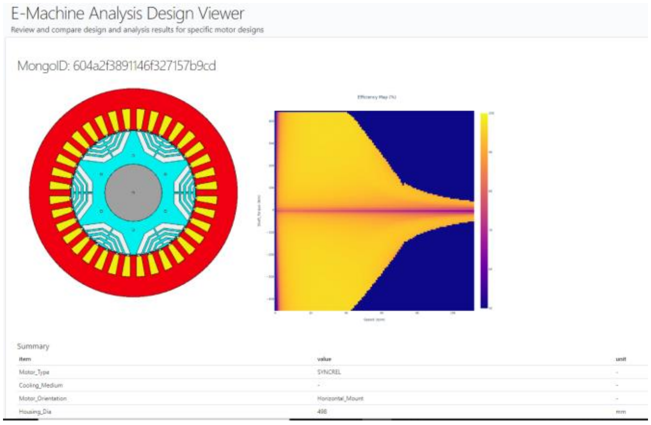
Following the literature review, the SMPM and SMR motor types were considered as the initial focus for the `EDU` library. Referring to Figure 16, within the `EmachineSummary` package, a standard interface called `MotorInterface` was implemented as a `partial` model to define drive unit subsystem of the overall e-propulsion system. In doing so, a user may choose between the various motor types, *e.g.*, SMPM. Referring to Figure 17, three electrical pins (`pin_p`, `pin_p1`, `pin_p2`) are used to connect to a three-phase inverter. Motor voltage, current, torque and rotational speed signals connect through a Control Bus to the EDU controller via an external control bus. The torque output connects with a vehicle model (gear and drive model) using a rotational flange connector from the MSL. The thermal port releases heat due to the power losses from the motor, as explained in Section 2.3.3.

In the `SMPMMotorTest` and `SMRMotorTest` packages, the standard SMPM and SMR Modelica models were extended with the sub-component models based on outputs from MotorCAD. The modifications are mainly to improve the power loss equations (see the next section) to enhance correlation with MotorCAD data and to model the thermal behavior (see Section 2.3.3).

2.3.3 Power loss models

Firstly, using Ricardo's MotorCAD database material (*e.g.*, see Figure 18), the generic power loss equations in Modelica were re-parameterized for a specific MotorCAD design, including stator core losses, stator winding losses, rotor winding losses and permanent magnet losses. The parameterization of the equations was performed using the MATLAB curve fitting toolbox and optimized to minimize the error across the range of operating motor speed and torque values. To achieve a better correlation with MotorCAD's data, some equations such as those for stator core losses were extended, for example, to include dependency on shaft torque. As MSL models do not specify an equation for the rotor core losses (lossPowerRotorCore is set to zero by default), a custom binomial equation as a function of motor speed and torque was included.

Through the `ThermalAmbientSMPM` interface block provided by Modelica, the motor power losses are extracted, and the detailed node temperatures are fed into the state-space thermal model. Temperatures corresponding to MotorCAD nodes ID 12 and 31 (the location nearest to the stator winding and permanent magnet, respectively) are used as temperature feedback signals to the thermal interface block.



E-Machine Analysis Design Viewer

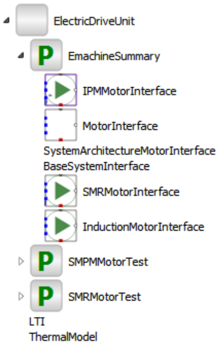
Review and compare design and analysis results for specific motor designs

MongoID: 604a2f3891146f327157b9cd

Summary

item	value	unit
Motor_Type	SYNCREL	-
Cooling_Medium	-	-
Motor_Orientation	Horizontal_Mount	-
Housing_Dia	498	mm

Figure 15. eMAD typical motor design



ElectricDriveUnit
EmachineSummary
IPMMotorInterface
MotorInterface
SystemArchitectureMotorInterface
BaseSystemInterface
SMRMotorInterface
InductionMotorInterface
SMPMMotorTest
SMRMotorTest
LTI
ThermalModel

Figure 16. Structure of `EDU` library

Table 2. Modelica imported parameters from MotorCAD

Parameters	*Value*	*Unit*
Number of poles in pairs	10	
Nominal frequency	600	Hz
Open circuit voltage	1000	v
Nominal stator resistance per phase	0.013	Ω
Stator main field inductance in d-axis	0.1755	mH
Stator main field inductance in q-axis	0.6618	mH
Stator copper loss	MotorCAD design map	
Stator iron loss	MotorCAD design map	
Magnet loss	MotorCAD design map	
Rotor iron loss	MotorCAD design map	

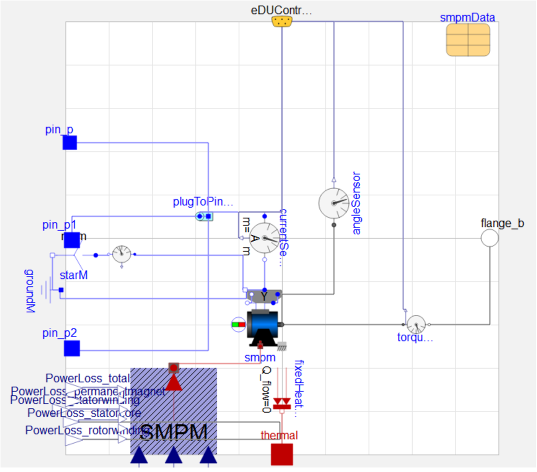


Figure 17. `IPM_motor_interface`

2.3.3 Thermal model

In order to better represent the thermal behavior of the EDU, a reduced order state-space thermal model (see Figure 19 and Figure 20) was also developed for Modelica and parameterized with MotorCAD data files, namely ".cmf" files for thermal capacitances, ".rmf" files for thermal resistances, ".pmf" files for power losses, ".tmf" files for node temperatures.

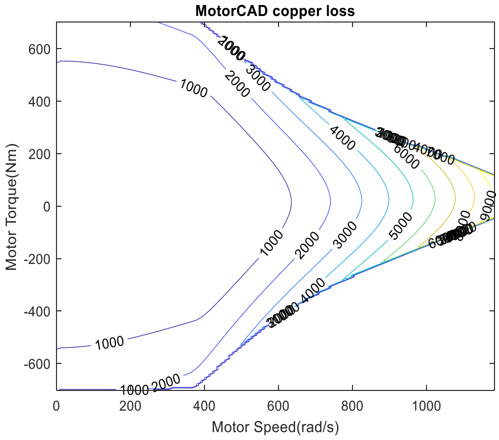
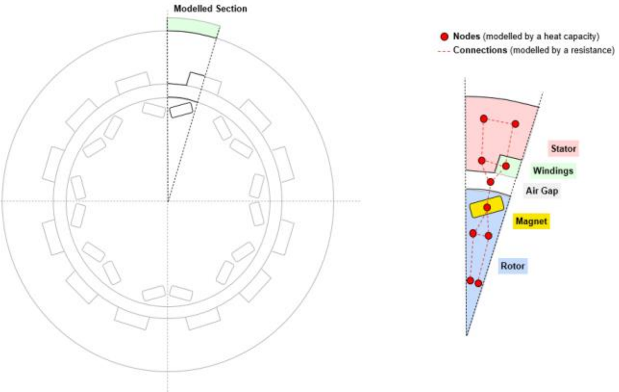
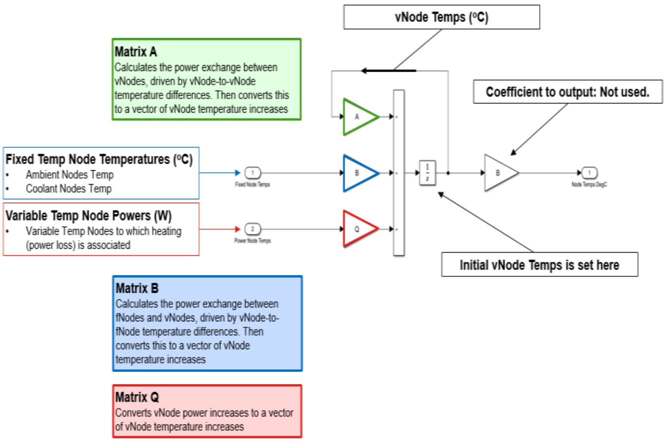


Figure 18. Example stator copper loss map from MotorCAD



Modelled Section
Nodes (modelled by a heat capacity)
Connections (modelled by a resistance)
Stator
Windings
Air Gap
Magnet
Rotor

Figure 19. A guide to reduced lumped-mass thermal models in MotorCAD



Matrix A
Calculates the power exchange between vNodes, driven by vNode-to-vNode temperature differences. Then converts this to a vector of vNode temperature increases

Fixed Temp Node Temperatures (°C)
- Ambient Nodes Temp
- Coolant Nodes Temp

Variable Temp Node Powers (W)
- Variable Temp Nodes to which heating (power loss) is associated

Matrix B
Calculates the power exchange between fNodes and vNodes, driven by vNode-to-fNode temperature differences. Then converts this to a vector of vNode temperature increases

Matrix Q
Converts vNode power increases to a vector of vNode temperature increases

Figure 20. Motor node temperature calculation

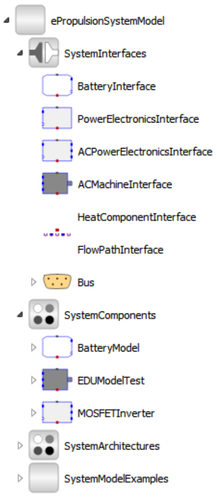


Figure 21. The structure of the `ePropulsionSystemModel`

2.4 `ePropulsionSystemModel`

The `ePropulsionSystemModel` library was built by using the components in `Battery`, `PowerElectronics` and `EDU` sub-libraries, and its structure is shown in Figure 21. The interfaces for the key components (`Battery`, `PowerElectronics` and `EDU`) in the `ePropulsionSystemModel` library were created in the `SystemInterfaces` package. Then the `replaceable` models of each of these components were built in the `SystemComponents` package by using the interfaces and the models developed in `Battery`, `PowerElectronics` and `EDU` sub-libraries. By connecting the `replaceable` `Battery` model, `EDU` model and `Inverter` model, an ePropulsion System Architecture system model can be created. Figure 22 shows an example model from the `SystemArchitectures` package which is an ePropulsion system model with Alternating Current (AC) architecture. Because the `Inverter` and `EDU` are `replaceable`, by choosing different `Inverter` and `EDU` models, different types of ePropulsion systems can be created. In this way, the ePropulsion system model can work as a standalone model or be integrated into complete vehicle models.

3 Use Case

To assess the ePropulsion system library, a vehicle co-simulation was done. The ePropulsion system library was imported into Ricardo IGNITE, which is a physics-based tool developed for complete vehicle system modelling and simulation. IGNITE features comprehensive built-in automotive Modelica libraries. These enable users not only to quickly and accurately model conventional and highly complex vehicle system models including hybrid-electric, full electric and novel vehicles, but also to import any Modelica based library such as the ePropulsion system library.

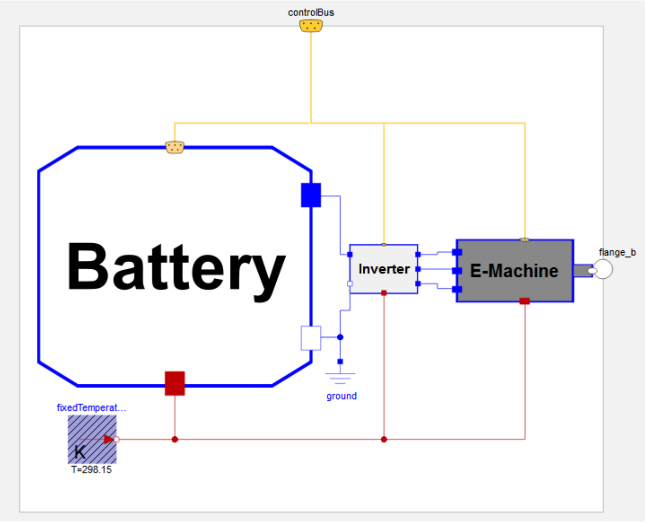


Figure 22. The ePropulsion system model with AC Architecture

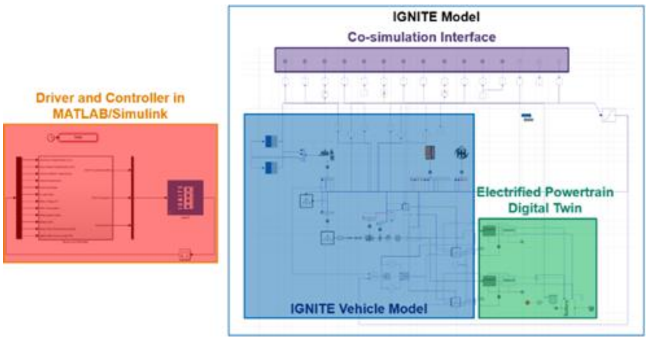

IGNITE Model
Co-simulation Interface
Driver and Controller in MATLAB/Simulink
Electrified Powertrain Digital Twin
IGNITE Vehicle Model

Figure 23. The co-simulation models

Figure 23 shows the co-simulation models used in this study. There are three important parts of the IGNITE model. These are the IGNITE vehicle model, the electrified powertrain digital twin and the co-simulation interface. The IGNITE vehicle model was built with the built-in, comprehensive vehicle modelling libraries. The electrified powertrain digital twin was built by using the components from the ePropulsion system library. The co-simulation interface provided the input and output interfaces for co-simulation. The vehicle controller used was developed in MATLAB/Simulink and coupled to the Modelica libraries, as shown in Figure 23, to showcase the ability to utilize pre-existing controllers developed in other environments. As this co-simulation model is a forward-facing simulation, a driver model was developed and included in MATLAB/Simulink as well.

The vehicle used in this study is a 6x4 long haul truck model which was defined based on typical MY2019 specifications. The key vehicle parameters are shown in Table 3. The vehicle performance attributes and propulsion system requirements are also given in Table 4.

Table 3. Vehicle parameters

Parameter	*Value*	*Unit*
GVW	44	t
Cd	0.6	-
Frontal area	10.2	m2
Tyre RRC	6.4	N/kN
Tyre rolling radius	0.49	m

Table 4. Vehicle performance attributes and propulsion system requirements

Vehicle Performance Attributes	*Propulsion System Requirements*
90km/h on 3% grade 40km/h on 10% grade Climbing 30% grade	Powertrain max continuous power ≥525kW
6x2 mode for GCW<32t (2nd axle lifted)	6x2 mode max continuous power ≥380kW
Min top speed 120km/h on flat road	Powertrain max continuous torque ≥67,500Nm ('at wheels')

A baseline configuration of the propulsion system was defined which has an 800 V system voltage, two EDUs, and one battery pack which has 530 kWh of nominal useable energy. To evaluate the baseline propulsion system performance, simulations were carried out using a variety of drive cycles.

Simulation Results

The drive cycle target speed profile is presented in Figure 24 alongside the actual vehicle speed. Using the imported controller and the Digital Twin models we are able to accurately match the actual speed with the target profile.

Having achieved the desired speed profile, we can then extract curves to assess the performance of each part of the electrified powertrain. Specifically, in Figure 25 the battery voltage, load current, delivered power and SoC are presented for the drive cycle. From those curves we can extract metrics such as depth of discharge and charge throughput for a single cycle, as well as total regeneration energy for that cycle.

In Figure 26 the inverter power losses and output phase currents are plotted, enabling us to evaluate if the semiconductor peak currents are within the absolute rating limits, as well as compare the inverter losses with the total delivered power by the battery.

Finally, Figure 27 and Figure 28 present curves extracted from a single eMachine model. In Figure 27 the eMachine torque and speed are plotted, which closely follow the controller demanded speed and torque, while in Figure 28 the different loss components, as well as the total losses of the eMachine are presented, enabling us to evaluate the performance of the machine.

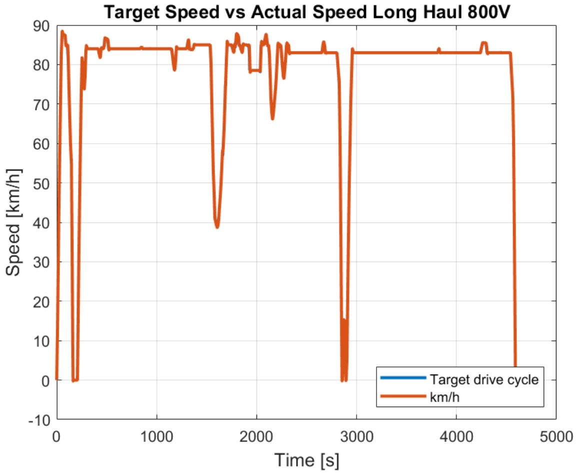


Figure 24. Target vs Actual Speed

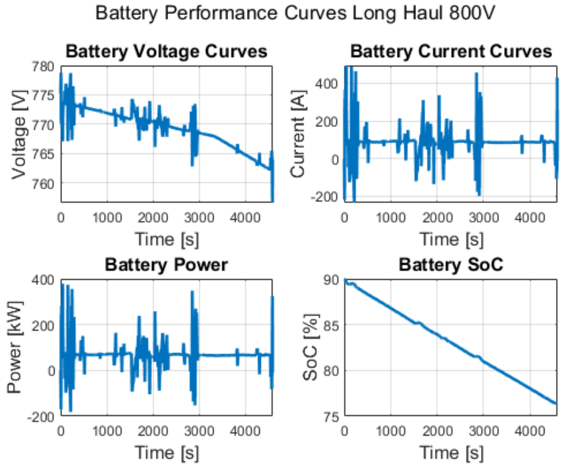


Figure 25. Battery Performance Curves

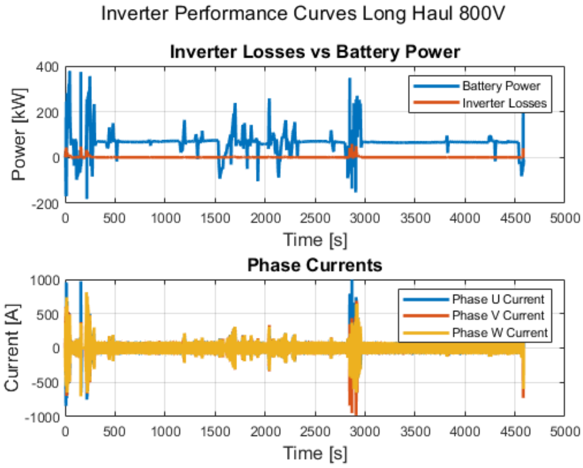


Figure 26. Inverter Performance Curves

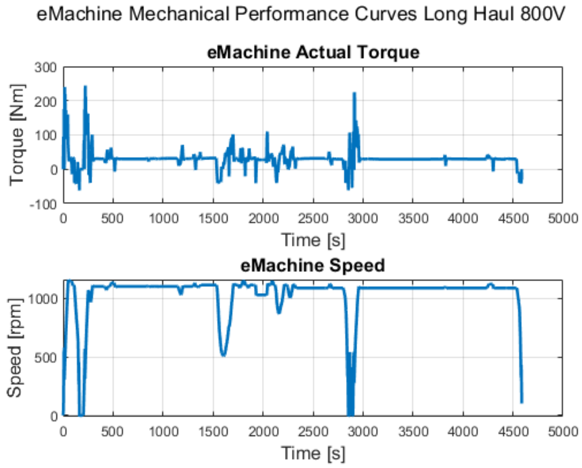


Figure 27. eMachine Mechanical Performance Curves

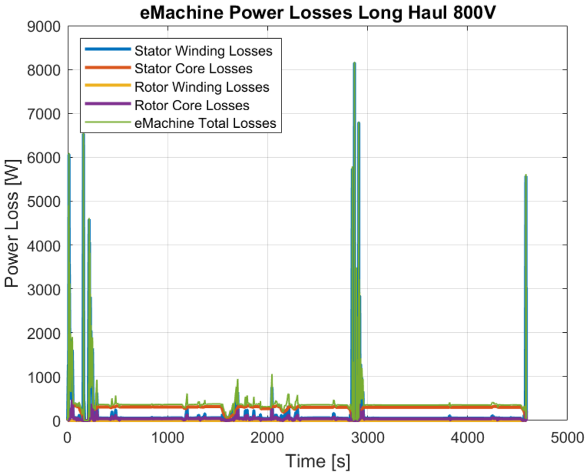


Figure 28. eMachine Power Losses

Total Cost of Ownership (TCO) Assessment

Optimising journey time and cost is the most important criteria for road-freight vehicles. To assess the impact of powertrain design and configuration on this criterion, a TCO study was carried out. The TCO study assessed the overall cost of vehicle ownership as €/t.100km and collected a wide range of inputs, both from the detailed system simulation previously discussed, expected degradation, component costs and costs of operation (electricity prices etc). The TCO model is presented schematically in Figure 29.

The following assumptions have been made for the purpose of the TCO assessment.

Table 5. TCO model assumptions

Parameter	*Value*	*Unit*
Electricity Price	0.25	€/kWh
Battery Pack Cost	200	€/kWh
Power Electronics and E-motor Cost	43	€/kWh

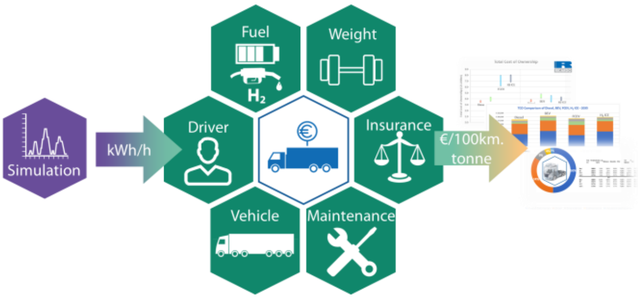


Figure 29. Schematic overview of Total Cost of Ownership model

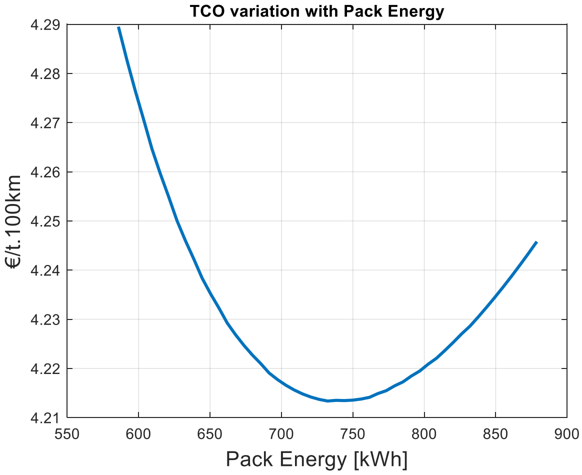


Figure 30. TCO sensitivity to battery pack energy

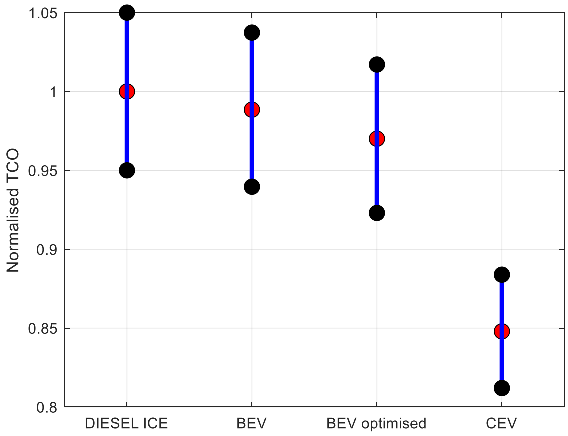


Figure 31. Comparison of TCO for different powertrain technologies

The TCO model enables a what-if analysis on the impact of various powertrain parameters (or indeed external parameters) on TCO. An example is shown in Figure 30 where the TCO sensitivity to battery pack installed energy is presented. A clear minimum can be seen which suggests that, for this vehicle application, it is beneficial to have a larger battery pack installed, which improves battery lifetime and range.

The TCO model is also used to compare different powertrain technologies as shown in Figure 31. In this example, the TCO for a Diesel Internal Combustion Engine (ICE) is used as a baseline and compared against a BEV, optimised BEV (using the initial results from above) and a Catenary Electric Vehicle (CEV) variant. A detailed comparison, including Fuel Cell EV will be the focus of a future study.

4 Conclusion

The Modelica library of electrified powertrain components, `ePropulsionSystem`, the sub-libraries of key sub-components (Battery, Power Electronics and EDU), and the various models included in the sub-libraries are detailed in the paper. The `ePropulsionSystem` library enables rapid construction of Digital Twins for Battery, EDU and Power Electronics systems in an electrified powertrain, and provides an approach which can scale automatically from low-fidelity for fast system level simulation to high-fidelity for sub-system design analysis in the electrified powertrain development process. An example of applying the `ePropulsionSystem` library to an EV use case is also discussed. In the use case, the co-simulation models and driving cycles simulation result are presented. Then, by using the co-simulation models, a Total Cost of Ownership (TCO) optimisation is discussed as an example to prove that the developed Modelica library can be used to assess and optimise a wide range of electrified propulsion architectures.

Acknowledgements

This work has been supported by Innovate UK (Project No: 81341). The authors would also like to thank Milan Cvetkovic and Dmitry Rozhdestvenskiy for their support with implementation of the software.

References

AG, I. T. (2020). *Transient thermal measurements and thermal.* Munich, Germany: Infineon Technologies AG.

Andrea Cordoba-Arenas, S. O. (2015). "A control-oriented lithium-ion battery pack model for plug-in hybrid electric vehicle cycle-life studies and system design with consideration of health management". In: *Journal of Power Sources*, Vol. 279, pp. 791-808.

Bao, R., Fotias, N., & McGahan, P. (2021). "Using Virtual Product Development with Design of Experiments to Design Battery Packs for Electrified Powertrain". In: SA*E Technical Paper 2021-01-0764*.

Bazzi, A. M. (2013). "Electric machines and energy storage technologies in EVs and HEVs for over a century". In: *International Electric Machines & Drives Conference.* Chicago, IL, USA: IEEE.

Cellier, F. E., Clauß, C., & Urquía, A. (2007). "Electronic circuit modeling and simulation in Modelica". In: *Proc. 6th EUROSIM Congress on Modelling and Simulation.* Ljubljana, Slovenia.

Ceraolo, M. (2015). "A new Modelica Electric and Hybrid Power Trains library". In: *Proceedings of the 11th International Modelica Conference, Versailles, September 21-23* (pp. 785-794). France: Linköping Electronic Conference Proceedings.

Chen, M., & Rincon-Mora, G. A. (2006). "Accurate electrical battery model capable of predicting runtime and I-V performance". In: *IEEE Transactions on Energy Conversion*, Vol: 21(2), pp. 504-511.

Dao, T.-S., & Schmitke, C. (2015). "Developing Mathematical Models of Batteries in Modelica for Energy Storage Applications". In: *The 11th International Modelica Conference.*

Denz, P., Schmitt, T., & Andres, M. (2014). "Behavioral Modeling of Power Semiconductors in Modelica". In: *Proceedings of the 10th International ModelicaConference*, (S. 343-352). Lund, Sweden.

Dorrell, D. G., Knight, A. M., Popescu, M., Evans, L., & Staton, D. A. (2010). "Comparison of different motor design drives for hybrid electric vehicle". In: *IEEE Energy Conversion Congress and Exposition.* Atlanta, GA, USA: IEEE.

Einhorn, M., Conte, F. V., Kral, C., Niklas, C., Popp, H., & Fleig, J. (2011). "A Modelica Library for Simulation of Elecric Energy Storages". In: *The 8th International Modelica Conference.*

Einhorn, M., F.V.Conte, C.Kral, C.Niklas, H.Popp, & J.Fleig. (2111). "A Modelica Library for Simulation of Electric Energy Storages". In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd;* (pp. 436-445). Dresden: Linköping University Electronic Press; Linköpings universitet.

Fan, G., Pan, K., Bartlett, A., Canova, M., & Rizzoni, G. (2014). "Electrochemical-Thermal Modeling of Li-Ion Battery Packs". In: *ASME 2014 Dynamic Systems and Control Conference.* San Antonio, Texas, USA.

Gerl, J., Janczyk, L., Krüger, I., & Modrow, N. (2014). "A Modelica Based Lithium Ion Battery Model". In: *The 10th International ModelicaConference.*

Gragger, J. V., Kral, H. G., Hansjörg, T. B., & Pirker, K. F. (2006). "The SmartElectricDrives Library – Powerful Models for Fast".In: *Modelica Conference 2006 at arsenal research in Austria.* Vienna: The Modelica Association .

Guo, M., Jin, X., & White, R. E. (2017). "An Adaptive Reduced-Order-Modeling Approach for Simulating Real-Time Performances of Li-Ion Battery Systems". In: *Journal of The Electrochemical Society*, Vol: 164(14), pp. A3602-A3613.

Haumer, A., & Kral, C. (2012). "Motor Management of Permagnent Magnet Synchronous Machines". In: *Proceedings of the 9th International Modelica Conference.* Munich: Modelica.

Haumer, A., Kral, C., Kapeller, H., Bäuml, T., & Gragger, J. V. (2009). "The Advanced Machines Library". In: *Proceedings 7th Modelica Conference, Italy, Sep. 20-22.* Como: Modelica.

He, H., Xiong, R., & Fan, J. (2011). "Evaluation of Lithium-Ion Battery Equivalent Circuit Models for State of Charge Estimation by an Experimental Approach". In *Energies* , Vol: 4(4), pp. 582-598.

Hwang, M.-H., Han, J.-H., Kim, D.-H., & Cha, H.-R. (2018). "Design and Analysis of Rotor Shapes for IPM Motors in EV Power Traction Platforms". *Energies*, vol: 2601.

Johnson, V. H., Pesaran, A. A., & Sack, T. (2000). "Temperature- Dependent Battery Models for High-Power

Lithium-Ion Batteries". In: *17th Annual Electric Vehicle Symposium*. Montreal, Canada.

Kral, C., & Haumer, A. (2011). "The New FundamentalWave Library for Modeling Rotating Electrical Three Phase Machines". In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy* (pp. 170-179). Dresden: Linköping University Electronic Press; Linköpings universitet.

Lai, S. C., Hill, C. I., & Suchato, N. (2019). "Implementation of an Advanced Modelica Library for Evaluation of Inverter Loss Modeling". In: *2019 IEEE Texas Power and Energy Conference (TPEC).* College Station, TX, USA.

Lee, B. H., Kim, K.-S., Jung, J.-W., Hong, J. P., & Kim, Y. K. (2012). "Temerature Estimation of IPMSM using Thermal Equivalent Circult". In: *IEEE Transactions of magnetics*, Vol. 48, No.11.

McDonald, D. (2012). "Electric Vehicle Drive Simulation with MATLAB/Simulink". In: *proceedings of the 2012 North-Central Section Conference.* American Society for Engineering Education .

McGahan, P., Rouaud, C., & Booker, M. (2019). "Comparison of Model Order Reduction Techniques for Real-Time Battery Thermal Modelling". In: *SAE Technical Paper 2019-01-0503*. doi:10.4271/2019-01-0503

Mohd, T. A., Hassan, M. K., & Aziz, W. M. (2015). "Mathematical modeling and simulation of an electric vehicle". In: *Journal of Mechanical Engineering and Sciences (JMES),* pp. 1312-1321.

Nelson, P., Dees, D., Amine, K., & Henriksen, G. (2002). "Modeling thermal management of lithium-ion PNGV batteries". In: *Journal of Power Sources,* Vol: 110(2), pp.349-356.

Park, C., & Jaura, A. K. (2003). "Dynamic Thermal Model of Li-Ion Battery for Predictive Behavior in Hybrid and Fuel Cell Vehicles". In: *SAE Technical Paper 2003-01-2286*. doi:10.4271/2003-01-2286

Perez, H., Shahmohammadhamedani, N., & Moura, S. (2015). "Enhanced Performance of Li-Ion Batteries via Modified Reference Governors and Electrochemical Models". In: *IEEE/ASME Transactions on Mechatronics, 20*(4), 1511-1520.

Pesaran, A. A. (2002). "Battery thermal models for hybrid vehicle simulations". In: *Journal of Power Sources,* Vol: 110(2), pp. 377-382.

Qin, D., Li, J., Wang, T., & Zhang, D. (2019). "Modeling and Simulating a Battery for an Electric Vehicle Based on Modelica". In: *Automotive Innovation,* Vol: 2(3), pp. 169-177.

Ramakrishnan, K., Stipetic, S., Gobbi, M., & Mastinu, G. (2018). "Optimal Sizing of Traction Motors Using Scalable Electric Machine Model". In: *IEEE Transactions on Transportation Electrification*, Vol: 4 , no. 1.

Surewaard, E., Karden, E., & Tiller, M. (2003). "Advanced Electric Storage System Modeling in Modelica". In: *The 3rd International Modelica Conference.*

Urkizu, J., Mazuela, M., Alacano, A., Aizpuru, I., Chakraborty, S., Hegazy, O., Klink, R. (2019). "Electric Vehicle Inverter Electro-Thermal Models Oriented to Simulation Speed and Accuracy Multi-Objective Targets". In: *Energies*.

Vetter, J., Novák, P., Wagner, M. R., Veit, C., Möller, K.-C., Besenhard, J. O., Hammouche, A. (2005). "Ageing mechanisms in lithium-ion batteries". In: *Journal of Power Sources*, Vol: 147(1-2), pp. 269-281.

Xing, Sun, J., & Lei. (2014). "Parameterization of Three-Phase Electric Machines Models for EMI sIMULATION". In: *IEEE Transactions on power electronics*, Vol: 29 , No. 1.

Development of a real-time test bed for indoor climate simulation in a VR environment using a digital twin

Kushagra Mathur[1] Christoph Nytsch-Geusen [1]

Lucas Westermann[1]

[1]Institute of Architecture and Urban Planning,
Berlin University of the Arts, Germany,
k.mathur@udk-berlin.de, nytsch@udk-berlin.de,
l.westermann@udk-berlin.de

Abstract

This paper describes the development process of a test bed for an interactive VR (virtual reality) environment for indoor climate simulations of buildings. The basic idea is to reproduce the simulated indoor climate of a thermal room model in a climate chamber with the help of air conditioning devices and thus to make the indoor climate directly physically experienceable for a user in real-time. In a first step, the real test bed is mapped with the help of a digital twin and simulated in parallel with the room model. In a second step, the digital twin is replaced by the real test bed and the Modelica room model is included then as an embedded model. In this way, the real test bed can be operated with the control algorithm which has been evaluated and optimized in a previous step. The described approach is demonstrated in a case study using a simple single-zone building model.

Keywords: hardware in the loop simulation, digital twin of a test bed, interactive virtual reality environment

1 Introduction

The traditional workflow in thermal building and indoor climate simulation consists of the creation of a thermal building model with simulation programs such as IDA ICE (https://www.equa.se/de/ida-ice) or EnergyPlus (https://energyplus.net), in which geometry, construction, and boundary conditions like the user behavior or outdoor climate of the real building or room are represented. As a result, the user receives time series of physical quantities describing the indoor climate, such as the air temperature, the radiation temperature, the humidity or the thermal comfort index PMV/PPD according to Fanger from these programs. In this way of working, simulation results are perceived by the user purely intellectually in the form of diagrams on a monitor.

However, since the indoor climate is primarily perceived via the physical senses, an immersive simulation process in which the user can physically experience the calculated indoor climate can lead to a deeper understanding of simulation scenarios. This approach is applied in the research project GEnEff, where the indoor climate simulated with a Modelica based thermal room model is reproduced in real-time with the help of a climate chamber and can thus be directly perceived by the user. The users are also visually immersed in the simulation process by using a VR environment, in which they can interact with the 3D representation of the room model. This includes, for example, opening windows or adapting the thermostat to different values of the set heating or cooling temperature, which influence the room climate. The feasibility of this simulation approach is described by the authors in Nytsch-Geusen et al., 2017 as well as its first realization in a corresponding test bed with thermal feedback for users in Nytsch-Geusen et al., 2021.

In this paper, we will show how the development process could be supported using a Modelica based digital twin that first evaluates the thermal, hydraulic and functional properties of the test bed before it is realized with constructional effort.

2 Related work

The Modelica modeling language has been used in various domains to recreate parts of technical systems in test benches as a model utilizing hardware in the loop (HIL) simulations. For example, Winkler and Gühmann, 2006 describe a real-time Modelica model of an engine test bench in which the real engine is embedded in a virtual test environment. Schneider et al., 2015 presents the HIL model of a test bench for investigating the dynamic behavior of circulation pumps, where the environment consisting of the energy system technology and the building envelope is virtually replicated in a Modelica model. Baltzer et al. 2014 describe a HIL test bench in which a real heat exchanger device using the waste heat of a car engine was coupled with a virtual thermal Modelica model of the heated vehicle cabin.

3 Methodology

Figure 1 describes the two-step research approach. In the first step, the climate chamber is represented by a Modelica model as a digital twin of the test bed, which contains both its thermal envelope and the air conditioning devices used to reproduce the calculated room climate.

DOI
10.3384/ecp21181263

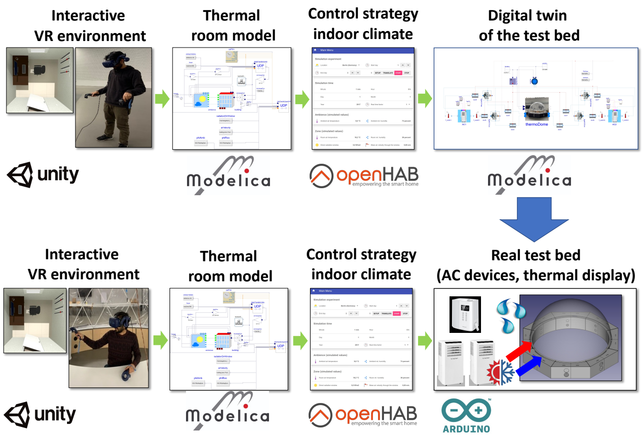

Interactive VR environment
Thermal room model
Control strategy indoor climate
Digital twin of the test bed
unity
Modelica
openHAB
Modelica
Interactive VR environment
Thermal room model
Control strategy indoor climate
Real test bed (AC devices, thermal display)
unity
Modelica
openHAB
ARDUINO

Figure 1. General research approach.

The user interacts via a VR headset in real-time with a 3D representation of the room model, which is implemented in Unity (Unity, 2021). The VR model is connected to the thermal room model also modeled in Modelica. The physical states calculated in this physical room model are sent to the event-based framework openHAB (openHAB, 2021). There, the room climate of the building model is compared with that of the climate chamber of the digital twin, and then a control algorithm is used to adjust the air conditioning devices in a manner that the differences between the two room climates are minimized. In this way, possible system configurations and control algorithms of the test bed can be evaluated in advance by making adjustments in the digital twin model. In the second step, the pre-optimized digital twin is replaced by the real test bed.

3.1 Control system with openHAB

openHAB (openHAB, 2021) is an open-source home automation software framework with a wide range of plugins connecting it to different hardware and software ecosystems. It uses an event-based bus to enable logical calculations or comparisons between the states of connected systems and devices as shown in Figure 2. This event bus can be manipulated and read by a REST API (Representational State Transfer Application Program Interface). For our purposes, two Python scripts access this data interface and are responsible for receiving and sending data via UDP (User Datagram Protocol) between Modelica and openHAB. This kind of data exchange was developed and evaluated by the authors in a case study for a digital twin, in which the control strategy of the building energy system for a research building was analyzed (Nytsch-Geusen et al., 2018).

Now, openHAB is used as the control system for both the test bed and its digital twin model. It performs all the logical calculations, does the comparisons of the simulated model states with the real sensor data of the climate chamber using a rule-based system. This controls the air conditioning devices of the test bed using the serial binding which then sends signals to Arduino micro-controllers. These rules are event-based logic algorithms which can perform different tasks when the conditions are triggered.

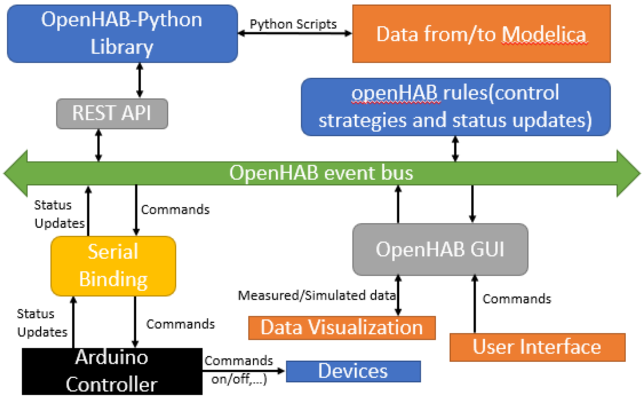


Figure 2. Interactions between openHAB event bus and connected systems and devices.

openHAB also delivers a centralized web interface to observe relevant data of the test bed and the simulation model and support the manipulation of the hardware system by the users in a standalone program without the need of receiving input data from Modelica or Unity.

3.2 Digital twin model

The comprehensive Modelica model of the digital twin includes different sub-models and objects interconnected with each other using distinct interfaces (compare with Figure 3). All used component models are based on or are included in the BuildingSystems library (Nytsch-Geusen et al., 2016) or the Modelica standard library (https://github.com/modelica/ModelicaStandardLibrary). This Modelica model exchanges data bi-directionally in real-time with both external software tools Unity (considering the user behavior in the VR environment) and openHAB (control logic of the climate chamber). For this purpose, a UDP data exchange model is responsible for sending or receiving data packages through UDP sockets to Unity and openHAB. This model class is implemented based on the Modelica_DeviceDrivers library (Thiele et al., 2017). It can consider also an adaptable real-time factor for increasing or decreasing the speed of the simulation experiment, depending on the present system dynamics or the presence of user interaction.

The upper part of Figure 3 shows the thermal building model for calculating the indoor climate, which receives its boundary conditions from an environment model in which different climate locations can be defined. On the other hand, the user's interactions with the visible 3D building model in the virtual reality environment are made available to the thermal building model via an interface to Unity. This can represent, for example, the user's presence in the room, opening a window, turning on the

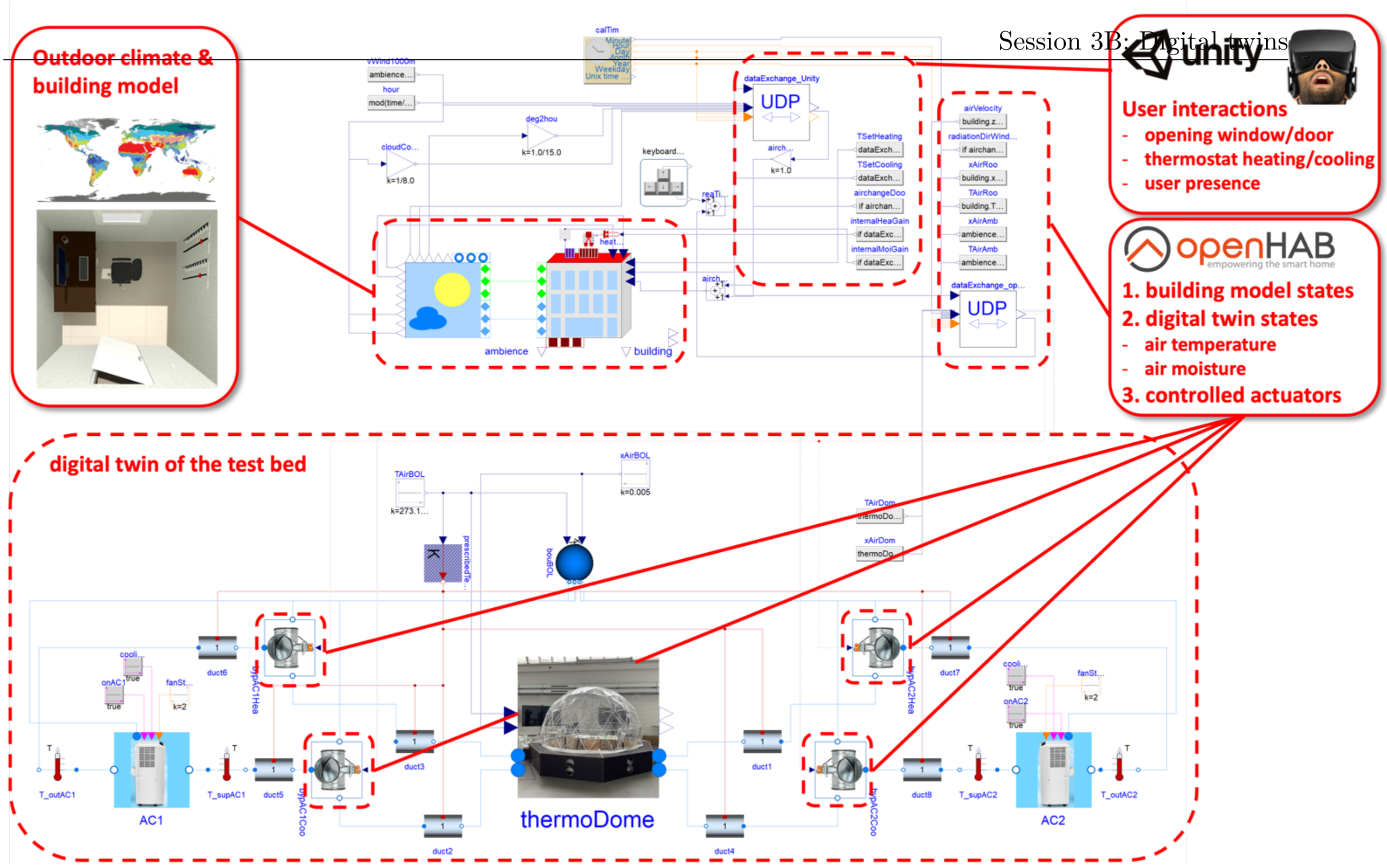


Figure 3. Digital Twin model of the real-time test bed.

lights, or changing the set values for the room thermostat. All these user interactions influence the indoor climate and are considered accordingly as dynamic boundary conditions in the thermal building model.

The lower part of Figure 3shows the sub-model of the test bed. This part of the system model was created keeping in mind the fact that this model can be used to design the real test bed. In the center, the thermal model of the climate chamber (thermoDome) is located, which is supplied with warm or cold air via a hydraulic air duct system using two air conditioning units. The hydraulic air duct system is part of the climate chamber model and it consists of two air ring tubes, one for cold air and one for warm air. Each of the air ring tubes is modeled as a detailed thermo-hydraulic model with two air inlets for the air conditioning devices and eight outlets to the inner space of the climate chamber (compare 4 with Figure 5).

Switching between hot and cold air supply is achieved via four bypass valves, which receive the control signals from the control logic of openHAB. It compares the room air temperature of the building model T_{Sim} with the air temperature of the climate chamber model T_{CC}. This is realized via a hysteresis algorithm, using the room air temperature of the building model as the target temperature. The cooling fans supply cold air to the climate chamber if the air temperature of the climate chamber model is higher than 1 K above the target temperature and closes all values when the temperature of chamber is 0.4 K to provide a buffer for stabilising the temperature as shown in the flowchart in Figure 5. Under this threshold, both the hot and cold bypass valves are closed. Similarly, the logic opens the heating valves if the threshold of 1 K below the room air temperature of the building model is reached.

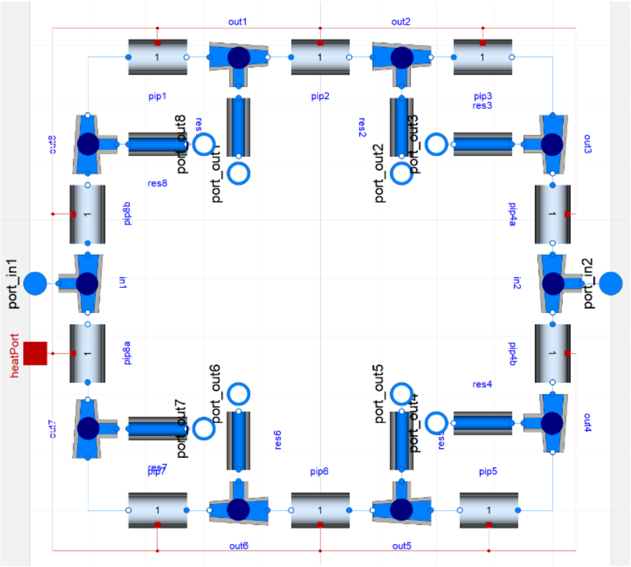


Figure 4. Thermo-hydraulic model of one of the two ring tubes which supply the climate chamber with warm or cold air through eight openings.

3.3 Real-time test bed

After simulating and testing the digital twin model, the real-time test bed was designed and built. In this context, more adaptations were made in the digital twin model and the test bed depending on the available hardware and their functioning.

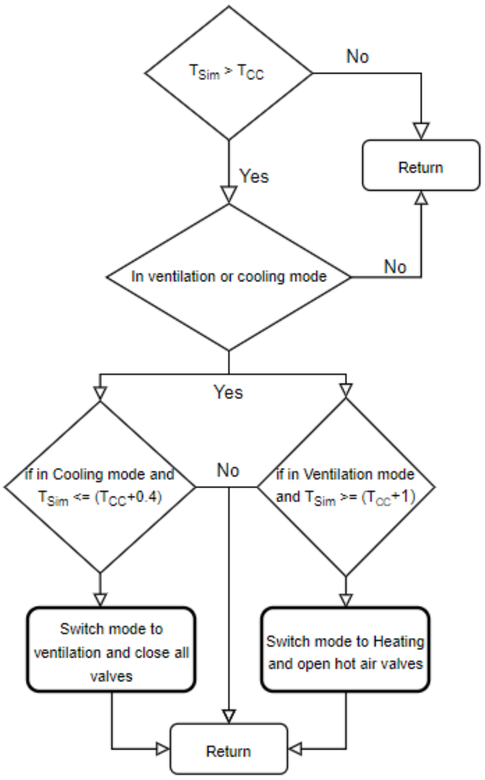

T_Sim > T_CC
No
Return
Yes
In ventilation or cooling mode
No
Yes
if in Cooling mode and T_Sim <= (T_CC+0.4)
No
if in Ventilation mode and T_Sim >= (T_CC+1)
Switch mode to ventilation and close all valves
Switch mode to Heating and open hot air valves
Return

Figure 5. A flowchart depicting one of the control algorithm used for checking and changing mode of the device.

The real-time test bed consists of an igloo-shaped tent and a base level construction (see Figure 4). The inner height of the dome is 2.8 m, and it has an inner diameter of 3.6 m. Two air duct ring tubes in the base level construction supply the inner space of the dome with warm and cold air through 16 circular openings from two air conditioning (AC) devices (compare with Figure 4 and Figure 7). The test bed includes two air conditioners for supplying the dome. These devices have a cooling capacity of 2,050 W, a heating capacity of 1,800 W, and a max. air volume flow rate of 320 m^3/h. In addition, two additional convective tower heaters can support a fast increase of the air temperature with a max. common capacity of 4,400 W. Two humidifiers can increase the air moisture level in the dome. The felt air movement through windows and doors can be simulated by two fans with 26 stages (max. airspeed 2.7 m/s). Finally, the felt solar radiation through closed or opened windows is simulated by an electric heating radiator in three stages with 850 W, 1,650 W, and 2,500 W.

All the hardware can be operated by infrared remote control. These infrared signals were decoded using the irlib2 library (`https://github.com/cyborg5/IRLib2`) for Arduino and the same library was used to encode it. Hence, these devices were controlled by an Arduino program. Since single-hose air-conditioning devices are used, it gives the advantage that, if the device is running in cooling mode supplying cold air, the exhaust port will supply warm air.

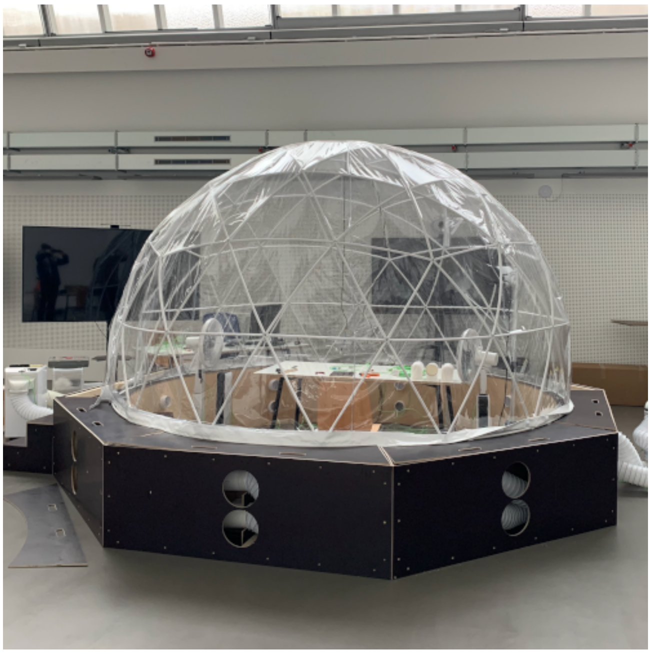


Figure 4. Real-time best bed with igloo-shaped climate chamber.

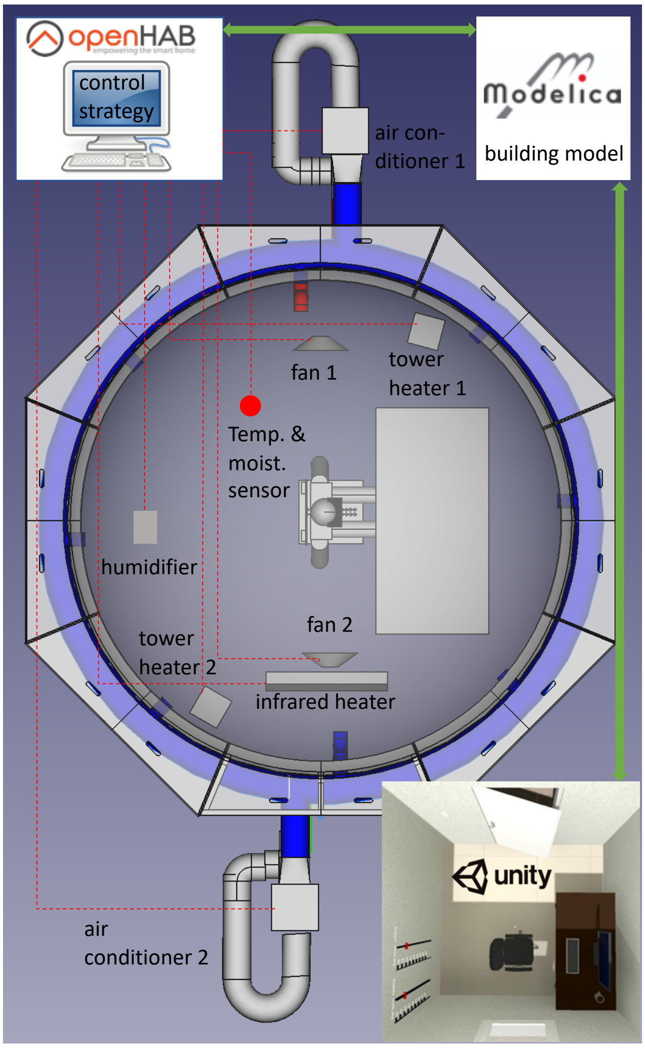


Figure 5. Configuration of the real-time test bed.

Because of this advantage, the device does not have to switch from heating to cooling mode which takes up to ten minutes. For that purpose, bypass valves were used to switch the airflow from the AC devices. The valves were also controlled by an Arduino using relays. The Arduino receives commands from openHAB through the serial port. Reusing the same rules as for the digital twin of the test bed, the air temperature inside the climate chamber can be manipulated.

3.4 Embedded building model

Figure 6 shows the Modelica thermal building model embedded in the test bed. In this case, it includes only the thermal room model and the UDP interface models to Unity and openHAB. It will be running in parallel to the real-time usage of the climate chamber and provides the necessary data for controlling and manipulating the indoor climate (air temperature, air moisture, air velocity through open windows, solar radiation through transparent façade areas, …) inside of the igloo. Certain parameters in the Modelica model can be kept dynamic, which can change with time and simulation will adapt accordingly. Such parameters include the location of the user (e.g., presence or absence in the room), the window opening state, or the set temperatures for heating and cooling.

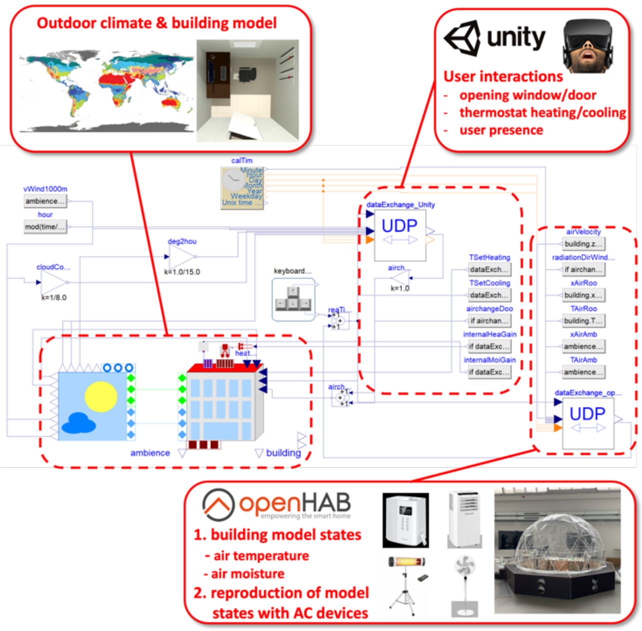


Figure 6. Embedded Modelica thermal building model.

4 Case study

The case study demonstrates the two-step research approach with the Modelica digital twin model and the Modelica HIL model. The simulation scenario takes place on the 1st of January at the climate location in El Gouna (Egypt) over a real-time period of 4,600 seconds. A simple cubic shaped one zone building model is used with a dimension of 3 m x 3 m x 3 m. It consists of four 20 cm concrete walls with outer (1.5 cm) and inner (2 cm) plaster, a 3 cm thick door, and a window with a U-value of 3.0 W/m^{2}K and g-value of 0.8. The floor and roof are constructed from 30 cm thick concrete. For adapting the indoor climate, the building model is equipped with an ideal heater and an ideal cooler, each with a max. capacity of 2,000 W. The set temperature for heating and cooling can vary in a range between 16 °C and 32 °C.

The scenario starts with an initialization temperature for the room model of 24 °C, both for the enclosed room air and for all component layers of the wall, floor, and ceiling models. Initially, a set cooling temperature of 32 °C and a set heating temperature of 18 °C are defined by the user in the VR environment, so that neither heating nor cooling is required in the room in the beginning of the experiment. The ambient temperature of the room model is approx. 20 °C during the experiment with a slight downward trend. After 20 minutes, the set cooling temperature is reduced from 24 °C to 20 °C by the user. From the 30th minute, the set heating temperature is then raised in several 2 °C steps every 10 minutes until it reaches 32 °C (for the digital twin) or 30 °C (for the real test bed) at the end of the experiment. The set cooling temperature is raised to 32 °C during the heating phase to avoid simultaneous heating and cooling in the building model.

5 Results

5.1 Digital twin model

Figure 7 shows the dynamic behavior of the thermal building model and the indoor climate of the climate chamber model. It can be seen that the calculated air temperature of the indoor model in the climate chamber can be reproduced well with the help of both air conditioning devices and the control strategy for the bypass dampers implemented in openHAB. The ambient temperature of the climate chamber model was set to 24 °C, corresponding to the ambient conditions of the laboratory in which the real test bed is located. The peaks of cooling and heating power are visible, which are caused by the user due to the change of the set temperatures.

5.2 Real-Time test bed

Figure 8 above shows the simulated indoor air temperature of the thermal building model and its replication in the real climate chamber. In this case, the calculated air temperature can be replicated less well in the climate chamber than with its digital twin over the entire period of the experiment. Qualitatively, the indoor climate is correctly reproduced, but the real climate chamber has difficulties in reproducing fast dynamic changes of the air temperature. Overall, the multi-stage heating-up process is better reproduced than the cooling-down process. The number of heating and cooling periods in the real experiment is significantly lower but they durations significant longer compared to the experiment with the digital twin.

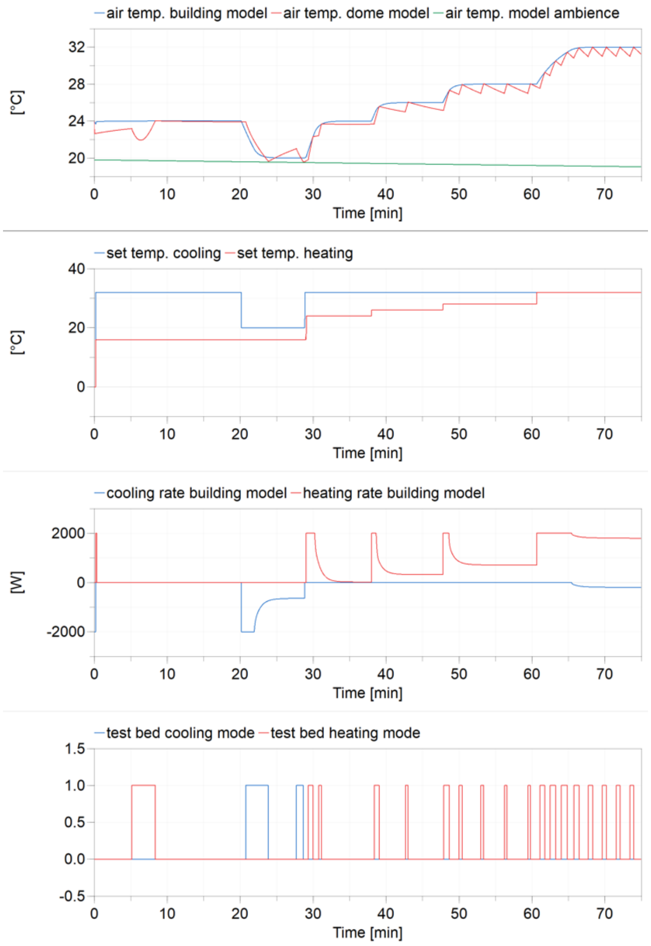


Figure 7. Results for the digital twin model.

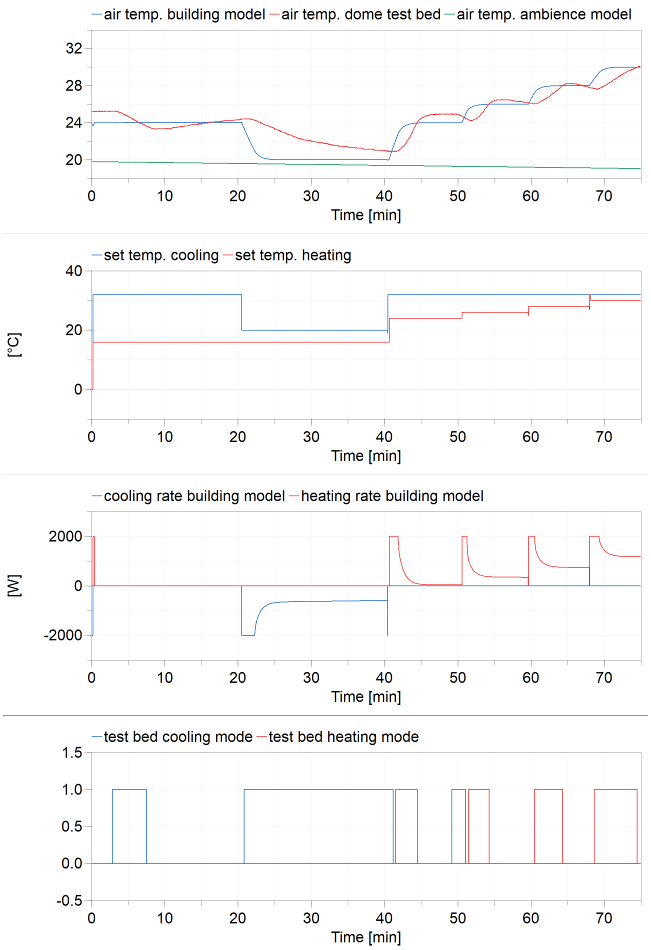


Figure 8. Results for the real-time test bed with the embedded building model.

6 Discussion

The case study has shown that the design of the test bed and the evaluation of associated control strategies can be tested in advance with the use of a digital twin. This knowledge can then be profitably transferred to the real test bed. It seems important to increase the cooling and the heating capacity in the climatic chamber to be able to simulate fast changes in the indoor climate sufficiently well in the real climatic chamber. This also includes a further development of the present discrete control strategy for the heating and cooling case, which has been kept very simple up to now. The use of continuous controllers or model-based controllers can be expected to provide better control quality. Also, a model extension of the digital twin seems to be necessary, which is partially still modeled too idealized. For example, performance-reducing leakages in the air distribution system are neglected, which cannot be completely prevented in the real test bed.

7 Summary and outlook

A two-step method is described and demonstrated in a case study to effectively support the development process of an interactive virtual reality environment for real-time simulation of indoor climate.

With the help of a digital twin of the real test bed, weak points in the experimental setup can be figured out in the simulation analysis. Furthermore, control algorithms can be tested and optimized efficiently in time by using an adaptive real-time factor.

The next development steps for the climate chamber will focus on the improvement of the achievable system dynamics. For this purpose, the power for heating and cooling has to be significantly increased and the control algorithms of the climate chamber has to be further developed.

Acknowledgments

The described research was conducted within the project “EnOB: GEnEff - Neuartige Bewertung der Gebäude-Energie-Effizienz und innovative Demonstration mittels Simulationsmethoden und Virtual Reality” and funded by the Federal Ministry for Economic Affairs and Energy in Germany (reference: 03EN1017A and 03EN1017B).

References

Baltzer, Sidney, Lichius, Thomas, Gissing, Jörg, Jeck, Peter, Eckstein, Lutz (2014). Hardware-in-the-Loop (HIL) Simulation with Modelica - A Design Tool for Thermal Management Systems. Proceedings from 10th International Modelica Conference. Lund (Sweden), 10-12 September 2014.

Nytsch-Geusen, Christoph, Banhardt, Christoph., Inderfurth. Alexander, Mucha, Katharina, Möckel, Jens, Rädler, Jörg, Thorade, Matthis and Tugores, Carles (2016). BuildingSystems – Eine modular hierarchische Modell-Bibliothek zur energetischen Gebäude- und Anlagensimulation. Proceedings from Conference BAUSIM 2016. Dresden (Germany), 14-16 September 2016.

Nytsch-Geusen, Christoph, Ayubi, Thaeba, Möckel, Jens, Rädler, Jörg and Thorade, Matthis (2017). BuildingSystems_VR – A new approach for immersive and interactive building energy simulation. Proceedings from Building Simulation 2017. San Francisco (USA), 7-9 August 2017.

Nytsch-Geusen, Christoph, Kaul, Werner, Kharraz, Sina (2018). Der digitale Zwilling in der energetischen Gebäude-und Anlagensimulation. Conference Proceedings from BAUSIM 2018. Karlsruhe (Germany), 26.-28. September 2018.

Nytsch-Geusen, Christoph and Mathur, Kushagra (2020). Entwicklung einer Virtual Reality-Umgebung zur interaktiven thermischen Raumsimulation. Bauphysik 42(6), 315-325.

Nytsch-Geusen, Christoph, Kaul, Werner, Mathur, Kushagra, Westermann, Lucas and Kriegel, Martin (2021). Development of an interactive virtual reality simulation environment with a thermal feedback for the user. Proceedings from Building Simulation 2021. Bruges (Belgium), 1-3 September 2021.

OpenHAB (2021). Official web site of OpenHAB: https://openhab.org (last access on 2021 May).

Schneider, Georg Ferdinand, Oppermann, Jens, Constantin, Ana, Streblow, Rita, Müller, Dirk (2015). Hardware-in-the-Loop-Simulation of a Building Energy and Control System to Investigate Circulating Pump Control Using Modelica. Proceedings from 11th International Modelica Conference. Versailles (France), 21-23 September 2015.

Thiele, Bernhard, Beutlich, Thomas, Waurich, Volker, Sjölund, Martin and Bellmann, Tobias (2017). Towards a Standard-Conform, Platform-Generic and Feature-Rich Modelica Device Drivers Library. Proceedings from 12th International Modelica Conference. Prague (Tschechien), 15–17 May 2017.

Unity (2021). Official web site of Unity: https://unity3d.com (last access on 2021 May).

Winkler, Dietmar and Gühmann, Clemens (2006). Synchronising a Modelica Real-Time Simulation Model with a Highly Dynamic Engine Test-Bench System. Proceedings from 4th International Modelica Conference. Vienna (Austria), 4-5 September 2006.

A first principles thermal losses model of the TCP-100 parabolic trough collector based on the Modelica Standard Library

Julia Pérez[1] Luis J. Yebra[1] Francisco M. Márquez[2,3] Pedro J. Zufiria[3]

[1]Plataforma Solar de Almería, CIEMAT, 04200 Tabernas, Spain
perezruizjulia@gmail.com,luis.yebra@psa.es
[2]Dpto. de Automática, Universidad de Alcalá (UAH), 28801 Alcalá de Henares, Spain,
francisco.marquez@uah.es
[3]Dpto. Matemática aplicada a las TIC, Information Processing and Telecommunications Center, ETSI Telecomunicación, Universidad Politécnica de Madrid (UPM), 28040 Madrid, Spain,
francisco.mgarcia@alumnos.upm.es,pedro.zufiria@upm.es

Abstract

The TCP-100 parabolic trough collectors (PTC) research facility at Plataforma Solar de Almería (CIEMAT) has been specially designed for the development of research activities in Automatic Control of PTC solar thermal power plants. The development of advanced control techniques for this kind of facilities requires dynamic models that should be successfully used in advanced controllers. An important part of these models is the thermal losses submodel, that traditionally has been considered as an experimental steady state correlation. In this paper, a work in progress about a first principles based model of the losses to the environment of a parabolic trough collector is presented, based on the physical phenomena inside any parabolic trough collector during the operation of the TCP-100 solar field. Concerning this model, the main contribution of the paper is to include the dependence of the air velocity close to the PTC. The implementation of the model in the Modelica language has been done prioritizing the use of the Modelica Standard Library classes. Some simulations results of this model with theoretical parameters values under typical operating conditions of the TCP-100 plant are presented, showing higher losses to the environment when compared with the information provided by the TCP-100 manufacturer.

Keywords: parabolic trough collector, thermal losses, object oriented modelling, Modelica Standard Library

1 Introduction

The main research facility at Plataforma Solar de Almería (PSA, www.psa.es) to test models and control algorithms has been the ACUREX parabolic trough collector (PTC) plant throughout its 32 years of life. Modelling and Control has been a research line at PSA under which mathematical models and advanced control techniques have been designed, implemented and tested. The ACUREX was replaced by the TCP-100 facility, shown in Figure 1, that presents important structural differences with respect to its predecessor, as outlined in section 2.

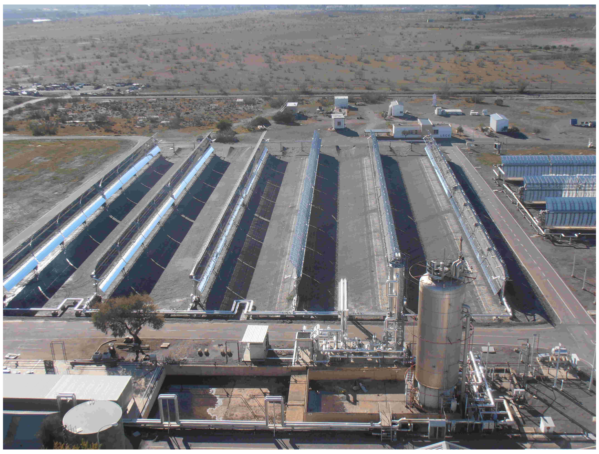


Figure 1. The TCP100 PTC solar research facility at Plataforma Solar de Almería (PSA-CIEMAT), (Pérez et al. 2018).

So far, modelling works based on the TCP-100 facility are: (Gallego, Yebra, Camacho, and Sánchez 2016), where a non-linear distributed parameter model of the TCP-100 solar field is presented; and (Pérez et al. 2018) in which the authors present a first principles system level dynamic model based on the project engineering data of the plant, complemented with several simulation experiments covering several design operation modes. Regarding control research activities, the only currently published work is (Gallego, Yebra, and Camacho 2018) where a Gain Scheduling Model Predictive Controller based on the model in (Gallego, Yebra, Camacho, and Sánchez 2016) is presented. In (Yebra, Márquez, and Zufiria 2020) a hybrid model of the TCP-100 facility is presented, covering the different operation modes of the plant and proving its applicability for operation training purposes. This hybrid model is implemented using both the StateGraph formalism in the Modelica object-oriented modelling language, and the Dymola tool. Currently, port-Hamiltonian modeling of multiphysics systems and object-oriented implementation techniques (Márquez, Zufiria, and Yebra 2020) are being applied to the modelling of the TCP-100

DOI
10.3384/ecp21181271

facility.

Usually, the losses model of a PTC has been approximated by a static nonlinear correlation-based model depending on the environment variables. Although the losses model presented here is more detailed than the other models used for control purposes, it still has low computational requirements, this fact allowing an efficient use of the model when it is connected to the other parts of the collector model. This model could be employed for the estimation of the losses and diagnosis based on its predictions independently in real-time applications during the operation of the plant, even in solar plants with a large number of collectors. We will test this model in the TCP-100 research plant to estimate each PTC power lost in real time.

The remainder of this article is organized as follows. In section 2 an introduction of the TCP-100 facility is presented, including the PTCs to which the model applies. In section 3 the model is presented, and in section 4 the implementation of the model with the Modelica Standard Library (MSL) is outlined. In section 5 different simulation results of the model are presented; finally, some conclusions are drawn in section 6.

2 The TCP-100 research plant

The plant is composed of two thermofluid circuits with different heat transfer fluids (HTFs) in each of them: Syltherm 800 in the solar field (primary circuit) and Therminol 55 in the storage part (secondary circuit). Both circuits are connected thermally by a heat exchanger (HEX). Figure 1 shows the complete facility where both parts (solar field and storage) are shown. In comparison to the ACUREX field, there are structural changes concerning the tanks system: there is now a new buffer tank T-2 in the primary circuit of 10 m^3 before the solar field pump. The previous tank T-1 of 115 m^3 has been maintained in the storage part (secondary circuit). The speed of the pumps for each of the two circuits can be controlled by the internal control loops inside variable frequency drives (VFD) connected to both pumps. An air cooler in the secondary circuit with a VFD has been installed to vary the cooling power in the storage part (the ACUREX field did not allow this). These are some of the new and significant structural changes of the solar thermal facility, that will require the application of more sophisticated operation modes. A diagram of the plant is shown in Figure 2.

2.1 The TCP-100 primary circuit

The TCP-100 primary circuit is mainly constituted by the solar field that occupies most of the plot, as it can be seen in Figure 1. The solar field is composed of three loops, each one with two PTCs in a North-South orientation. Each PTC is 100 m long, and it is formed by 8 modules connected in series. Figure 3 shows the first PTC of the first loop, numbered from the inlet (and coolest part) of the loop.

The PTCs in each loop are connected in the South extreme, and the coldest PTC will be always the first in the row, placed at the right part of each loop in Figure 1. Each of them has two PTCs connected in series, ordered from 1 (rightmost) to 6 (leftmost). The first loop is formed by the connected pair 1st-2nd (right loop), the second one by 3rd-4th (center loop) and the third one by 5th-6th (left loop).

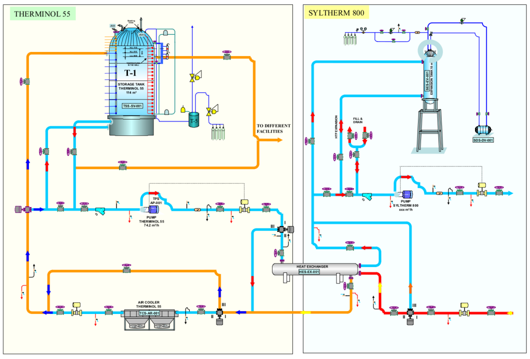


Figure 2. Diagram of the TCP-100 plant showing the main components in both subcircuits: primary (Syltherm800) and secondary (Therminol 55).

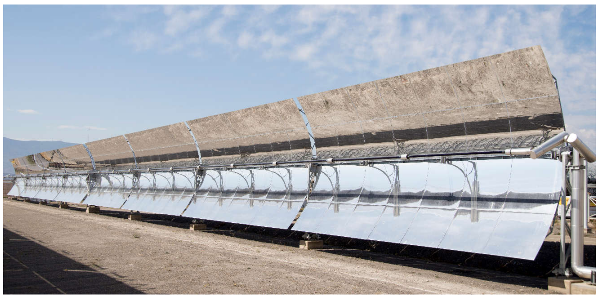


Figure 3. First PTC of the first loop of the TCP-100 solar field, (Pérez et al. 2018).

The thermal losses (sub)model of a PTC plays an important role in any solar thermal plant model. It will be applied to all of the PTCs in any solar thermal facility. In our experimental facility TCP-100 we have 6 PTC units but an industrial plant could contain thousands of them. This model has an important influence on the efficiency of the plant operation and has a marked importance in the design of advanced control strategies. This fact has motivated our work on deriving an independent submodel based on first principles for the thermal losses in a PTC, that should be directly connected to other model parts of the collector; or it could also be used independently as a standalone model.

The technology of a parabolic trough collector is described with detail in (Zarza 2003), where the components involved in the losses to the environment are presented.

3 TCP-100 parabolic trough collector thermal losses model

The model that describes the heat losses is founded in well known physical phenomena: conduction, convection and radiation (Bejan 2016). With the objective of using this model independently of the other dynamics in a PTC, we will consider boundary conditions the temperature of the HTF inside the PTC absorber pipe, the ambient temperature T_{amb}, and the wind velocity close to the PTC absorber pipe. Although the solar irradiance is usually an important boundary condition in solar thermal models, it is not considered in this case since it has no direct implication in the presented losses model. The presented model is a first principles model, including correlations to approximate the solution of the heat transfer coefficients. The equations of conservation and the energy flows are implemented in the MSL classes.

Figure 4 shows a simplified representation of the heat flows in a PTC absorber pipe. In such figure, $\dot{Q}_{incident}$ and $\dot{Q}_{lost}$ represent, respectively, the heat flow rates incident into and lost away from the absorber pipe.

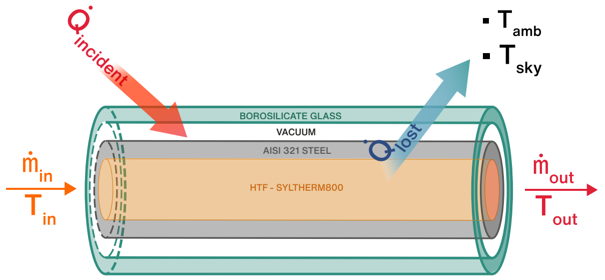


Figure 4. Heat flows in a PTC.

In Figure 5 a representation of the internal heat flow rates between the different sections is sketched.

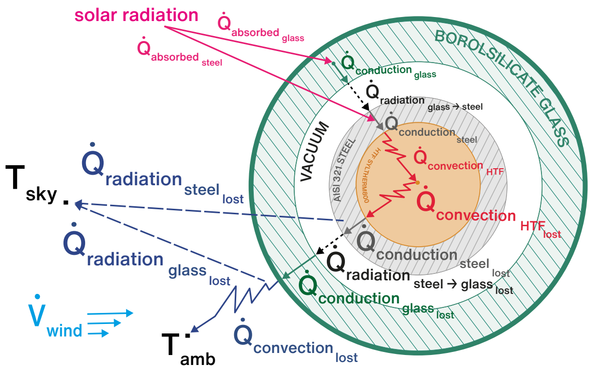


Figure 5. Heat flows in a section of a PTC.

The equations for the thermal losses model are detailed in (Wagner and Gilman 2011), (Pérez et al. 2018) and (Zarza 2003), and the flows have been organized for the implementation in the MSL as follows:

- $\dot{Q}_{absorbed,glass}$ is the heat flow absorbed by the envelope glass from the reflected concentrated solar radiation.
- $\dot{Q}_{absorbed,steel}$ is the heat flow that passed through the envelope glass and is absorbed in the pipe (steel).
- $\dot{Q}_{conduction,glass}$ is the conduction heat flow inside the glass envelope.
- $\dot{Q}_{radiation,glass-steel}$ is the radiation heat flow between the glass envelope and the pipe.
- $\dot{Q}_{conduction,steel}$ is the conduction heat flow inside the pipe.
- $\dot{Q}_{convection,\mathrm{HTF}}$ is the conduction-convection heat flow from the internal surface of the pipe to the HTF. The model used of the heat transfer coefficient (h_{HTF}) is based on the Dittus-Boelter correlation (Cengel 2014) with $n = 0.4$:

$$\begin{aligned} h_{\mathrm{HTF}} &= \frac{\mathrm{Nu}\cdot k_{\mathrm{HTF}}}{2\cdot r_0},\\ \mathrm{Nu} &= 0.023\cdot \mathrm{Re}^{0.28}\cdot \mathrm{Pr}^{0.4},\\ \mathrm{Re} &= \frac{\mathrm{v}_{\mathrm{HTF}}\cdot 2\cdot r_0\cdot \rho_{\mathrm{HTF}}}{\mu_{\mathrm{HTF}}},\\ \mathrm{Pr} &= \frac{c_{p,\mathrm{HTF}}\cdot \mu_{\mathrm{HTF}}}{k_{\mathrm{HTF}}}, \end{aligned} \tag{1}$$

where Nu, Re and Pr are respectively the Nusselt, Reynolds and Prandtl numbers; k_{HTF} is the themal conductivity of the HTF, r_0 the internal radius of the absorber pipe, $\mathrm{v}_{\mathrm{HTF}}$ is the mean HTF velocity, ρ_{HTF} is the density, $c_{p,\mathrm{HTF}}$ is the specific heat, and μ_{HTF} represents the dynamic viscosity.

- $\dot{Q}_{convection,lost}$ is the heat flow rate lost by convection to the environment. This flow depends on the environment temperature and the wind velocity close to the PTC, based on the correlation corresponding to the heat transfer coefficient defined by Churchill y Bernstein in (Cengel 2014):

$$\begin{aligned} h_{air} &= \frac{\mathrm{Nu}\cdot k_{air}}{2\cdot r_1},\\ \mathrm{Nu} &= 0.3 + \frac{0.62\cdot \mathrm{Re}^{\frac{1}{2}}\cdot \mathrm{Pr}^{\frac{1}{3}}}{\left[1+\left(\frac{0.4}{\mathrm{Pr}}\right)^{\frac{2}{3}}\right]^{\frac{1}{4}}}\cdot\left[1+\left(\frac{Re}{282000}\right)^{\frac{5}{8}}\right]^{\frac{4}{5}},\\ \mathrm{Re} &= \frac{\mathrm{v}_{wind}\cdot 2\cdot r_1\cdot \rho_{air}}{\mu_{air}},\\ \mathrm{Pr} &= \frac{c_{p,air}\cdot \mu_{air}}{k_{air}}, \end{aligned} \tag{2}$$

where all the variables have the same meaning as above and are evaluated for the air medium at ambient temperature T_{amb}. In particular, v_{wind} is the wind velocity close to the outlet radius of the glass envelope r_1.

- $\dot{Q}_{radiation,steel-lost}$ is the radiation heat flow lost to the environment passing through the envelope glass. $\dot{Q}_{radiation,glass-lost}$ is the radiation heat flow lost to the environment. For both heat flows the sky temperature approximation considered is the following (Zarza 2003)

$$T_{sky} = T_{amb} - 10. \qquad (3)$$

4 MSL implementation of the TCP-100 thermal losses model

The main objective in this section is to implement the model of the PTC reusing the classes of the MSL. For the thermal losses model we have instantiated mostly the `Modelica.Thermal.HeatTransfer` subpackage classes, according to the phenomena involved: conduction, convection, and radiation. Only the parameters and variables needed to define the heat transfer coefficients and/or the thermal conductances had to be added to our model. More details can be read in the MSL documentation (Fritzson 2014).

Figure 9 shows the thermal losses model obtained by properly instantiating and parameterizing the MSL components. The connector `port_a` (instance of `HeatPort_a`) is the point where the boundary condition representing the fluid temperature will be connected. In a complete model of a PTC, this temperature is usually a state variable or an algebraic variable computed from an equation of state for the HTF, and thus, it is assumed to be known. The flows in the model are all implemented with the classes `Modelica.Thermal.HeatTransfer .Components`. The energy conservation in the glass envelope and the metal pipe masses are represented by the class `HeatCapacitor`. In the cases where the thermal conductances in the conduction-convection and radiation flow models were variables instead of parameters, `RealExpression` blocks have been properly used to consider this. In order to define boundary conditions, the wind velocity close to the absorber pipe, and the ambient temperature are assumed to be known from experimental measurements.

The relation of the instances of MSL classes presented in Figure 9, that implement the first principles models of the heat flows (detailed in section 3) is as follows: $\dot{Q}_{absorbed,glass}$ in Q_incident_glass; $\dot{Q}_{absorbed,steel}$ in Q_steel_incident; $\dot{Q}_{conduction,glass}$ in Q_Conduction_glass1 and Q_Conduction_glass2; $\dot{Q}_{radiation,glass-steel}$ in Q_radiation_steel_glass; $\dot{Q}_{conduction,steel}$ in Q_conduction_steel1 and Q_conduction_steel2; and, $\dot{Q}_{convection,\mathrm{HTF}}$ is the convection heat flow from the pipe to the HTF, computed by Equation 1, considering the temperature difference between the boundary of the fluid `port_a.T` and the bulk fluid temperature (not represented in Figure 9).

The parts of the model not implemented in the components of the MSL have been implemented in Modelica code. For example, this is the case of all the variables and parameters associated with equations (1) to (3).

5 Simulation results of the TCP-100 thermal losses model

In this section we present the results of a simulation experiment of the Modelica losses model in Figure 9, performed with the Dymola tool.

In Figure 6 the boundary conditions applied to the model are shown: concentrated direct normal irradiance, ambient temperature, and wind velocity profile assumed close to the absorber pipe. In this case, the dynamics between the direct normal irradiance and the inlet HTF temperature have been included in order to illustrate the use of an experimental irradiance register from Plataforma Solar de Almería.

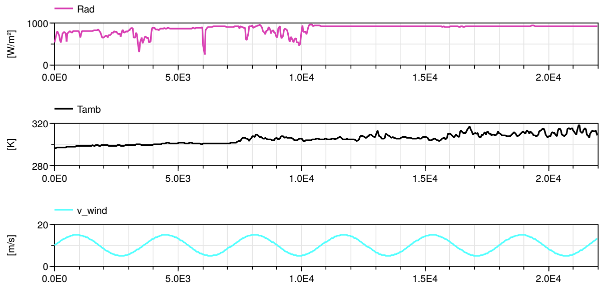


Figure 6. Boundary conditions in the simulation experiment. Direct normal solar irradiance (Rad), ambient temperature (T_amb), and wind velocity (v_wind).

In Figure 7, a comparison of the results of three different global losses model for the TCP-100 PTC is depicted. Q_corr presents the results of the model used in (Pérez et al. 2018) that is a global model assuming losses by convection and radiation from the glass envelope; Q_Pipe-maker presents the results obtained from a model based on the data specifications from the manufacturer of the TCP-100 PTC, and Q_Fpm shows the results of the first principles model presented in this article, as detailed in sections 3 and 4. In all three cases, the PTC model (excluding the thermal losses part) is the one presented in (Pérez et al. 2018). Although the simulation experiment shown in this figure contains nominal parameters for the model in Figure 9, calibration activities with experimental data in the TCP-100 facility might show some variations. It can be observed that the losses predicted by the model presented in this paper (Q_Fpm) are higher than those predicted by the manufacturer of the PTC (Q_Pipe-maker), and also higher than those predicted by the thermal model used in (Pérez et al. 2018). In the absence of the required experimental validation, our interpretation of this preliminary result is that the dependence of the wind velocity in the new proposed model could be producing higher thermal losses.

Figure 8 focuses on the presented first principles losses model, showing the different heat flow rate components under the boundary conditions profile provided in Figure 6. It is shown that the convection heat flow rate from the glass envelope (Qconv_lost) is the component with

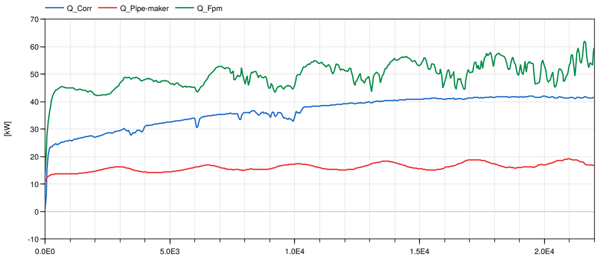


Figure 7. Simulation results of different thermal losses models: Q_corr presents the results of the model in (Pérez et al. 2018), Q_Pipe-maker presents the results obtained with a model based on the data specifications from the manufacturer of the TCP-100 PTC; and Q_Fpm represents the results of the first principles model presented in this article.

higher losses. The radiation losses from the steel pipe (Qrad_steel_lost) are lower than the convection losses from the glass envelope (Qconv_lost); also, the component with the lowest losses is Qrad_glass_lost, from the glass envelope, due to the lower mean temperature of this part.

The presented results are considered preliminary because the parameters used for the simulation of the proposed model have been obtained from data specifications from the PTC manufacturer and the bibliography. To validate these preliminary results, a set of experiments will be performed in the TCP-100 facility, with the objective of calibrating the parameters according to the experimental data to be obtained in future tests.

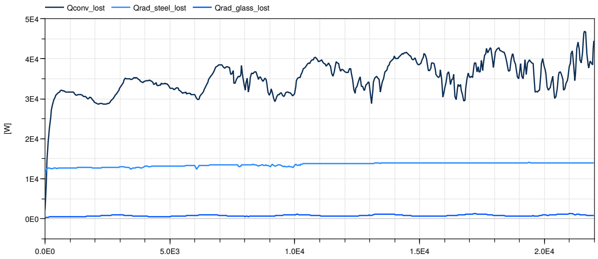


Figure 8. Simulation of the heat flow rates of thermal losses to the environment. Qconv_lost is the convection heat flow rate from the glass envelope, Qrad_steel_lost is the radiation heat flow rate from the steel pipe, and Qrad_glass_lost is the radiation heat flow rate from the glass envelope.

6 Conclusions

In this paper we have presented a thermal losses model of each of the PTCs in the TCP-100 solar themal power plant installed at Plataforma Solar de Almería (CIEMAT). This new model is based on first principles and includes an additional input variable with respect to the traditional approaches. The model is implemented with classes from the Modelica Standard Library (MSL), which have been complemented with the necessary equations to define the parameters and the variables required by the MSL, which have been obtained from diverse theoretical references. This is an ongoing work and the results presented are only preliminary, and they need to be validated with experimental data to be obtained in the TCP-100 facility.

Future works might include the results of the calibration of parameters in the presented first principles model, based on the experimental data from the TCP-100 research facility.

Acknowledgements

The work of Luis J. Yebra was supported by the Universidad de Alcalá, through the Programa propio Giner de los Ríos Research Grant.

References

Bejan, Adrian (2016). *Advanced engineering thermodynamics*. John Wiley & Sons.

Cengel, Yunus (2014). *Heat and mass transfer: fundamentals and applications*. McGraw-Hill Higher Education.

Fritzson, Peter (2014). *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. John Wiley & Sons.

Gallego, Antonio J., Luis J. Yebra, and Eduardo F. Camacho (2018). "Gain Scheduling Model Predictive Control of the New TCP-100 Parabolic Trough Field". In: *IFAC-PapersOnLine* 51.2, pp. 475–480. ISSN: 24058963. DOI: 10.1016/j.ifacol.2018.03.080. URL: https://doi.org/10.1016/j.ifacol.2018.03.080.

Gallego, Antonio J., Luis J. Yebra, Eduardo F. Camacho, and Adolfo J. Sánchez (2016). "Mathematical Modeling of the Parabolic Trough Collector Field of the TCP-100 Research Plant". In: *Proceedings of The 9th EUROSIM Congress on Modelling and Simulation, EUROSIM 2016, The 57th SIMS Conference on Simulation and Modelling SIMS 2016*, pp. 912–917. DOI: 10.3384/ecp17142. URL: https://doi.org/10.3384/ecp17142.

Márquez, Francisco M., Pedro J. Zufiria, and Luis J. Yebra (2020). "Port-Hamiltonian modeling of multiphysics systems and object-oriented implementation with the Modelica language". In: *IEEE Access* 8, pp. 105980–105996. DOI: 10.1109/access.2020.3000129. URL: https://ieeexplore.ieee.org/document/9110502.

Pérez, J et al. (2018). "First Principles System Level Modelling of TCP-100 Facility for Simulation of Operation Modes". In: *IFAC-PapersOnLine* 51.2, pp. 481–486. DOI: 10.1016/j.ifacol.2018.03.081. URL: https://doi.org/10.1016/j.ifacol.2018.03.081.

Wagner, Michael J. and Paul Gilman (2011). *Technical manual for the SAM physical trough model*. Vol. 303. June, pp. 275–3000. URL: http://www.nrel.gov/docs/fy11osti/51825.pdf.

Yebra, Luis J., Francisco M. Márquez, and Pedro J. Zufiria (2020). "Simulation of TCP-100 Facility System Level Model for Operation Training Purposes". In: *Proceedings of The 7th Annual Conf. on Computational Science & Computational Intelligence (CSCI'20)*. DOI: 10.1109/CSCI51800.2020.00251.

Zarza, Eduardo (2003). "Generación directa de vapor con colectores solares cilindro parabólicos Proyecto DIrect Solar Steam (DISS)". PhD thesis. Universidad de Sevilla, pp. 1–480. URL: https://idus.us.es/handle/11441/15300.

DOI
10.3384/ecp21181271

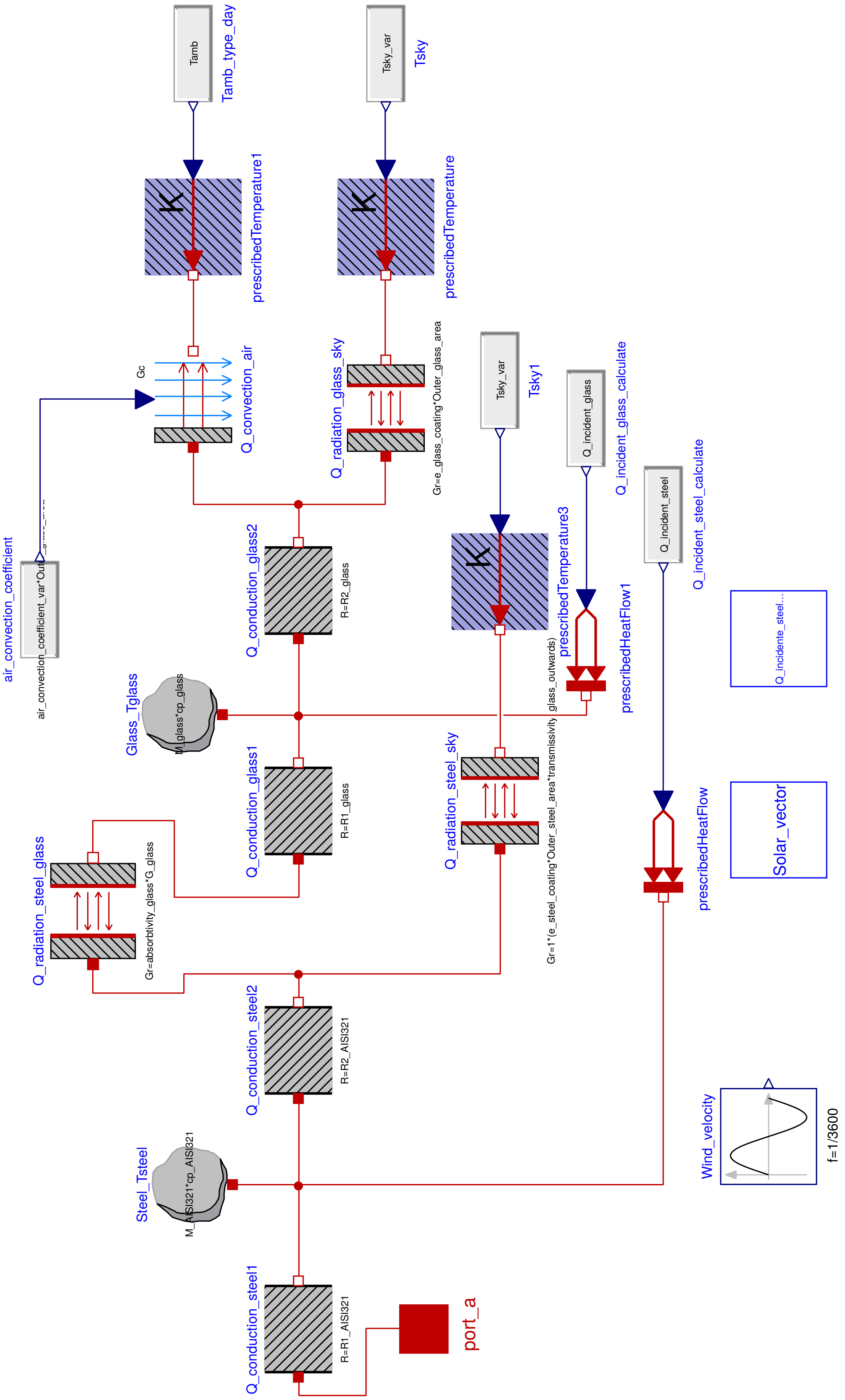


Figure 9. Modelica model of the thermal losses of the PTC TCP-100 implemented with the Modelica Standard Library.

Electromagnetic Transient Simulation of Large Power Networks with Modelica

Alireza Masoom[1] Jean Mahseredjian[1] Tarek Ould-Bachir[2] Adrien Guironnet[3]

[1]Department of Electrical Engineering, Polytechnique Montreal, Canada,
{alireza.masoom, jean.mahseredjian}@polymtl.ca
[2]Department of Computer Engineering, Polytechnique Montreal, Canada,
tarek.ould-bachir@polymtl.ca
[3]Réseau de Transport d'Électricité, France,
adrien.guironnet@rte-france.com

Abstract

This paper presents the simulation of electromagnetic transients (EMTs) with Modelica. The advantages and disadvantages are discussed. Simulation performance and accuracy are analyzed through the IEEE 118-bus benchmark which includes EMT-detailed models with nonlinearities. The domain-specific simulator EMTP is used for validations and comparisons.

Keywords: Modelica, Equation-based Modeling, Acausal modeling, Electromagnetic transients, EMT, Synchronous machines, Large scale, Nonlinearity, IEEE 118-bus grid.

1 Introduction

Power system simulations are mainly categorized into phasor-domain and time-domain. In phasor-domain analysis, voltages and currents are computed as phasors varying in time. The electrical network is solved in steady-state and at the fundamental frequency. Power generation sources, such as synchronous machines, are solved with their differential equations in time-domain and interfaced with network equations using phasor equivalents. The phasor-domain approach is suitable for electromechanical transients, load flow, and short-circuit studies in very large-scale grids and can be applied to any linear system but becomes less accurate with the presence of power electronics-based equipment e.g., FACTS and HVDC since this technique cannot represent the faster transients. Its main advantage remains computational speed for studying lower frequency transients, but harmonics and nonlinear models are ignored.

In the time-domain simulation approach, the network and all integrated components are solved in time-domain with detailed differential equations. Harmonics and nonlinearities are modeled accurately. The solution may include lower frequency interactions, as well as very high-frequency transients. The time-domain approach allows to solve networks for electromagnetic transients and is tagged as the EMT-type solution.

Various specialized EMT-type simulation tools (Mahseredjian 2009) are currently available and well adapted for studying various power system phenomena, including transformer saturation effects, lightning and switching transients, and integration of inverter-based resources. Power electronics-based components can be simulated very accurately.

The EMTP (Mahseredjian, et al. 2007) software used in this paper is widely used for the EMT simulations. The cornerstone of this software is the discretization of component models using the well-known companion circuit approach. The A-stable trapezoidal and Backward-Euler numerical integration methods are employed for the discretization. The latter is used at discontinuity points to avoid numerical oscillations (Sana, Mahseredjian, et al. 1995). In EMTP, the companion circuits of components are interconnected (to respect Kirchhoff's first law) through a sparse matrix solver using the Modified Augmented Nodal Analysis (Mahseredjian, Dennetière, et al. 2007) formulation.

In power system modeling, Modelica has been first considered for phasor-domain simulations. iTesla Power Systems Library (iPSL) (Vanfretti et al. 2016) is a comprehensive Modelica package that was generated through the iTesla project (Lemaitre 2014) for unified modeling and for facilitating network model exchanges amongst transmission system operators. PowerGrids (Bartolini et al. 2019) and ObjectStab (Larsson 2004) are other advanced libraries developed for electromechanical and stability analyses, respectively.

Modelica (Fritzson 2014) is an *equation-based* language that relies on the description of a system by differential-algebraic equations. The EMT behavior of electrical components can be modeled by their differential equations. The language emphasizes the *acausal* approach based on *declarative* modeling where the model and solver are decoupled. It significantly improves model development process and model readability.

The first attempt to develop a Modelica-based *EMT-detailed* simulator was reported in (Masoom et al. 2020), where transmission lines incorporating the constant parameter and wideband (Kocar Mahseredjian 2016) models were introduced and validated. An EMT-detailed library was developed and validated using the IEEE 39-bus network.

A challenging problem with Modelica is computational speed (including compilation and simulation time). Some

DOI
10.3384/ecp21181277

solutions are based on numerical optimizations, e.g., Jacobian optimization (Kofman, Fernández, and Marzorati 2021) and simulation in DAE mode (Braun et al. 2017; Henningsson 2019). An open-source solution (called Dynaωo) is specifically designed for power system simulations in phasor-domain (Guironnet et al. 2018). Dynaωo approach is based on hybrid coding with Modelica and C++ and demonstrates competitive performance compared to the specific domain packages (Guironnet et al. 2018). This approach was used for EMT simulations in (Masoom et al. 2021) as well. Even though it delivers better performance compared to the pure Modelica tools, there is still a significant gap in comparison with EMTP.

The IEEE 118-bus benchmark contains the following models: synchronous generators (including magnetic saturation model) with controls, transformers, transmission lines, nonlinear inductances, and nonlinear surge arresters. The basic models, such as resistance, inductance, and advanced models, that is, various models of transmission line, loads, saturable transformers, synchronous machine (without saturation), machine controls, etc. were already presented in previous papers (Masoom et al. 2020; Masoom et al. 2021). This paper focuses on the synchronous machine model with saturation and the nonlinear arrester and comparison of simulation efficiency in Modelica.

The paper is organized as follows. Two selected models, namely the synchronous machine and nonlinear arrester are developed using Modelica in Section 2. The numerical results for the IEEE 118-bus system are provided in Section 3.

2 Modelica Model Implementation

In this Section, two nonlinear models are presented and discussed in more detail. The models have been developed based on EMTP mathematical representations.

2.1 Synchronous Machine Model with Magnetic Saturation

The details required to model Synchronous Machine (SM) depend on the type of transient study. Saturation effects in SM are important for EMT analysis. Assuming the SM is modeled by two damping windings on the *q*-axis (denoted by *kq1*, *kq2*) and one damper winding (*kd*) and one field winding (*fd*) on the *d*-axis, (1)-(6) provide the flux-based equations of SM in *state-variable form.*

$$\mathbf{v}_{dq0} = \mathbf{P}(\theta)\mathbf{v}_{abc}/\mathrm{V}_{stator,base} \tag{1}$$

$$p\boldsymbol{\psi} = \omega_b(\mathbf{A}\boldsymbol{\psi} + \mathbf{u}) \tag{2}$$

$$\mathbf{A} = -(\mathbf{R}\mathbf{L}^{-1} + \mathbf{W}) \tag{3}$$

$$\mathbf{i} = \mathbf{L}^{-1}\boldsymbol{\psi} \tag{4}$$

$$\mathbf{i}_{abc} = \mathbf{P}^{-1}(\theta)\mathbf{i}_{dq0} \tag{5}$$

$$\mathbf{i}_{abc,actual} = \mathbf{i}_{abc}\cdot \mathrm{I}_{stator,base} \tag{6}$$

where:

$$\mathbf{u} = \left[\mathrm{v}_q, \mathrm{v}_d, \mathrm{v}_{fd}, 0, 0, 0\right]^T \tag{7}$$

$$\boldsymbol{\psi} = \left[\psi_q, \psi_d, \psi_{fd}, \psi_{kd}, \psi_{kq1}, \psi_{kq2}\right]^T \tag{8}$$

$$\mathbf{i} = \left[\mathrm{i}_q, \mathrm{i}_d, \mathrm{i}_{fd}, \mathrm{i}_{kd}, \mathrm{i}_{kq1}, \mathrm{i}_{kq2}\right]^T \tag{9}$$

$$\mathbf{i}_{dq0} = \left[-\mathrm{i}_q, -\mathrm{i}_d, 0\right]^T \tag{10}$$

$$\mathbf{R} = \mathrm{diag}\left(\mathrm{R}_a, \mathrm{R}_a, \mathrm{R}_{fd}, \mathrm{R}_{kd}, \mathrm{R}_{kq1}, \mathrm{R}_{kq2}\right) \tag{11}$$

In the above equations, the operator p is $\frac{d}{dt}$, the vector $\mathbf{v}_{abc}$ is the terminal voltage, $\mathbf{v}_{dq0}$ is the voltage in *dq* frame, ω_b is the base angular velocity, $\mathbf{P}(\theta)$ is the Park's transformation, vectors $\mathbf{u}$, $\mathbf{i}$, and $\boldsymbol{\psi}$ denote the stator and rotor voltages, currents, and flux linkages in the *dq* frame and $\mathbf{i}_{abc}$ is the stator current. $\mathbf{W}_{6\times6}$ is the rotor speed-dependent matrix; all elements are zero except $w[1,2] = \omega_r$ and $w[2,1] = -\omega_r$, $\mathbf{L}_{6\times6}$ is the symmetrical matrix of inductances in the rotor reference frame, $\mathbf{R}_{6\times6}$ is the stator and rotor resistance matrix.

The details of saturation effects modeling in the *dq* axes are explained in (Karaagac et al. 2011). In magnetic saturation modeling, the following assumptions are made: (1) The leakage flux saturation and cross saturation are ignored. It means only magnetizing inductances, L_{md} and L_{mq} are saturable. (2) Saturation is determined by the air-gap flux linkage. (3) The sinusoidal distribution of the magnetic field over the face of the pole is unaffected by saturation.

Since the saturation relationship between the total air-gap flux, ψ_T, and the magnetomotive force under loaded conditions is assumed to be the same as at no-load conditions, therefore magnetic saturation of stator and rotor iron can be modeled by the *no-load saturation curve* which is characterized by a *piecewise linear* graph (Karaagac et al. 2017).

Consequently, the mathematical model of saturation is introduced by:

$$\psi_T = f(\psi_{T,us}) = f\left(\sqrt{\psi_{md,us}^2 + \psi_{mq,us}^2}\right) \tag{12}$$

$$\begin{aligned}\psi_{md,us} &= \mathrm{L}_{md,us}\mathrm{i}_{md}\\ \mathrm{i}_{md} &= \mathrm{i}_d + \mathrm{i}_{fd} + \mathrm{i}_{kd}\end{aligned} \tag{13}$$

$$\begin{aligned}\psi_{mq,us} &= \mathrm{L}_{mq,us}\mathrm{i}_{mq}\\ \mathrm{i}_{mq} &= \mathrm{i}_q + \mathrm{i}_{kq1} + \mathrm{i}_{kq2}\end{aligned} \tag{14}$$

where $\psi_{T,us}$ is the total unsaturated air-gap flux, $\psi_{md,us}$ and $\psi_{mq,us}$ are the unsaturated magnetizing flux linkages, $\mathrm{L}_{md,us}$ and $\mathrm{L}_{mq,us}$ are the unsaturated magnetizing inductances, and i_{md} and i_{mq} are the magnetizing currents; each on the *dq* axis, respectively. Throughout the paper, the subscript *sat* and *us* mean saturated and unsaturated, respectively.

The value of saturated magnetizing flux linkages on the *dq* axis ($\psi_{md,sat}$ and $\psi_{md,sat}$) can be corrected by a ratio of corresponding unsaturated values as illustrated in Figure 1.a. In EMTP, the magnetic saturation is represented by a *piecewise linear* curve as sketched in Figure 1.b. For the *j*th operating segment, ψ_T is given by:

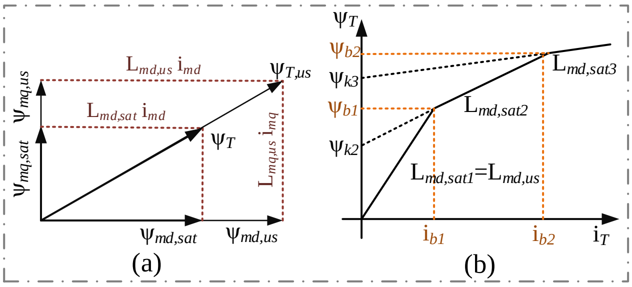


Figure 1. (a): Saturated and unsaturated magnetizing flux linkages in the *dq* axes of a synchronous machine. (b): Magnetic saturation characteristic (*piecewise-linear* approximation).

$$\begin{aligned}\psi_T &= \psi_{kj} + b_j \psi_{Tu} \\ &= \psi_{kj} + b_j L_{md,us} i_T\end{aligned} \tag{15}$$

$$i_T = \sqrt{i_{md}^2 + \left(\frac{L_{mq,us}}{L_{md,us}}\right)^2 i_{mq}^2} \tag{16}$$

$$b_j = \frac{L_{md,satj}}{L_{md,us}} \tag{17}$$

where b_j is the saturation factor and ψ_{kj} is the residual flux. The saturated values $L_{md,sat}$ and $L_{mq,sat}$ are computed as:

$$\begin{aligned}L_{md,sat} &= b_j L_{md,us} \\ L_{mq,sat} &= b_j L_{mq,us}\end{aligned} \tag{18}$$

For a *salient pole* machine, because of large airgap path along the *q*-axis, it is only required to correct the ψ_{md}; thus:

$$\begin{aligned}L_{md,sat} &= b_j L_{md,us} \\ L_{mq,sat} &= L_{mq,us}\end{aligned} \tag{19}$$

Figure 2 demonstrates the solution procedure for the electrical equations of SM. In the case of no saturation, the relationship between field current (i_{fd}) and terminal voltage (v_t) is linear; therefore, the magnetizing inductances in (20) are constant ($q_j = d_j = 1$). If saturation is selected, it is required to compute the magnetizing inductances at each time point; thus, $\mathbf{L}$ is time-variant ($q_j = d_j = b_j$ for round rotor and $q_j = 0$, $d_j = b_j$ for salient pole machine). This method results in implicit equations requiring an *iterative* solution.

The model discussed above has been implemented for the first time in Modelica. The model code is illustrated in Figure 3. The declaration of variables and the conversion of operational parameters to the standard ones are hidden to conserve space and only the `equation` section is demonstrated. The terminal voltages of SM are represented by `Pk.pin[1].v`, `Pk.pin[2].v` and `Pk.pin[3].v` for the phases *a*, *b* and *c*, respectively. `P(theta)` represents a pre-defined `function` for the Park's transformation calculations.

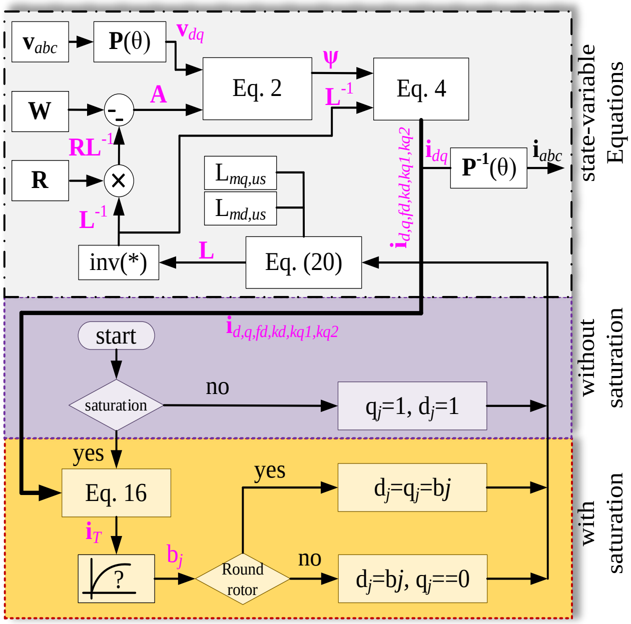


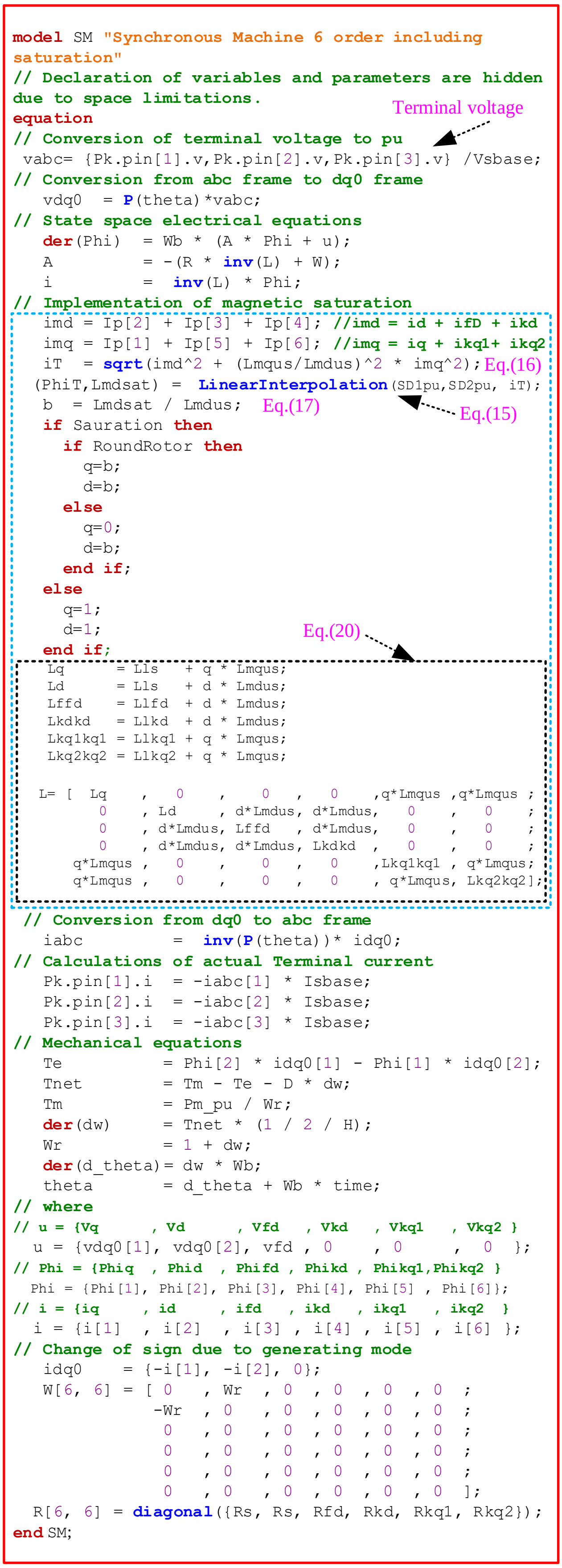
Figure 2. Solution procedure of synchronous machine with/without magnetic saturation in Modelica (Only electrical equations are demonstrated).

Equation (2) is used as a differential equation for the implemented model; the state vector `Phi` represents the flux linkages and the input vector `u` the voltages. The system matrix `A` is time-variant and computed as per (3).

The matrix of parameters for representation of saturation, `SD`, is given by a 2-by-*n* matrix, where *n* is the number of points taken from the no-load saturation curve. The first row of this matrix contains the values of field currents (physical value), while the second row contains values of corresponding terminal voltages (per unit). `LinearInterplate(SD1PU, SD2PU, iT)` is a `function` to interpolate the `iT` by the two vectors of field current (`SD1PU`) and voltage (`SD2PU`). These two vectors are calculated in the *non-reciprocal* per unit. The function returns the total flux (`PhiT`) and `Lmdsat` which the latter is used for calculation of coefficient `b` as per (17). The stator physical currents are represented by `Pk.pin[1].i`, `Pk.pin[2].i` and `Pk.pin[3].i` for the phases, *a*, *b* and *c*, respectively.

Other pieces of code represent the mechanical equations of SM which are not discussed in this paper.

$$\mathbf{L} = \begin{pmatrix} L_{ls} + q_j L_{mq,us} & 0 & 0 & 0 & q_j L_{mq,us} & q_j L_{mq,us} \\ 0 & L_{ls} + d_j L_{md,us} & d_j L_{md,us} & d_j L_{md,us} & 0 & 0 \\ 0 & d_j L_{md,us} & L_{lfd} + d_j L_{md,us} & d_j L_{md,us} & 0 & 0 \\ 0 & d_j L_{md,us} & d_j L_{md,us} & L_{lkd} + d_j L_{md,us} & 0 & 0 \\ q_j L_{mq,us} & 0 & 0 & 0 & L_{lkq1} + q_j L_{mq,us} & q_j L_{mq,us} \\ q_j L_{mq,us} & 0 & 0 & 0 & q_j L_{mq,us} & L_{lkq2} + q_j L_{mq,us} \end{pmatrix} \tag{20}$$



```
model SM "Synchronous Machine 6 order including
saturation"
// Declaration of variables and parameters are hidden
due to space limitations.
equation
// Conversion of terminal voltage to pu
 vabc= {Pk.pin[1].v,Pk.pin[2].v,Pk.pin[3].v} /Vsbase;
// Conversion from abc frame to dq0 frame
   vdq0   = P(theta)*vabc;
// State space electrical equations
   der(Phi)   = Wb * (A * Phi + u);
   A          = -(R * inv(L) + W);
   i          =  inv(L) * Phi;
// Implementation of magnetic saturation
   imd = Ip[2] + Ip[3] + Ip[4]; //imd = id + ifD + ikd
   imq = Ip[1] + Ip[5] + Ip[6]; //imq = iq + ikq1+ ikq2
   iT  = sqrt(imd^2 + (Lmqus/Lmdus)^2 * imq^2);
  (PhiT,Lmdsat) =  LinearInterpolation(SD1pu,SD2pu, iT);
   b  = Lmdsat / Lmdus;
   if Sauration then
     if RoundRotor then
       q=b;
       d=b;
     else
       q=0;
       d=b;
     end if;
   else
     q=1;
     d=1;
   end if;
    Lq       = Lls   + q * Lmqus;
    Ld       = Lls   + d * Lmdus;
    Lffd     = Llfd  + d * Lmdus;
    Lkdkd    = Llkd  + d * Lmdus;
    Lkq1kq1  = Llkq1 + q * Lmqus;
    Lkq2kq2  = Llkq2 + q * Lmqus;

  L= [  Lq     ,   0     ,    0    ,   0     ,q*Lmqus ,q*Lmqus ;
         0     , Ld      , d*Lmdus, d*Lmdus,    0    ,    0    ;
         0     , d*Lmdus, Lffd    , d*Lmdus,    0    ,    0    ;
         0     , d*Lmdus, d*Lmdus, Lkdkd   ,    0    ,    0    ;
      q*Lmqus ,   0     ,    0    ,   0     ,Lkq1kq1 , q*Lmqus;
      q*Lmqus ,   0     ,    0    ,   0     , q*Lmqus, Lkq2kq2];
  // Conversion from dq0 to abc frame
   iabc           =  inv(P(theta))* idq0;
// Calculations of actual Terminal current
   Pk.pin[1].i   = -iabc[1] * Isbase;
   Pk.pin[2].i   = -iabc[2] * Isbase;
   Pk.pin[3].i   = -iabc[3] * Isbase;
// Mechanical equations
   Te            = Phi[2] * idq0[1] - Phi[1] * idq0[2];
   Tnet          = Tm - Te - D * dw;
   Tm            = Pm_pu / Wr;
   der(dw)       = Tnet * (1 / 2 / H);
   Wr            = 1 + dw;
   der(d_theta)= dw * Wb;
   theta         = d_theta + Wb * time;
// where
// u = {Vq      , Vd      , Vfd   , Vkd   , Vkq1   , Vkq2 }
  u = {vdq0[1], vdq0[2], vfd , 0      , 0       ,  0   };
// Phi = {Phiq  , Phid  , Phifd , Phikd , Phikq1,Phikq2 }
  Phi = {Phi[1], Phi[2], Phi[3], Phi[4], Phi[5] , Phi[6]};
// i = {iq     , id     , ifd   , ikd   , ikq1   , ikq2  }
  i = {i[1]   , i[2]   , i[3] , i[4] , i[5] , i[6] };
// Change of sign due to generating mode
   idq0     = {-i[1], -i[2], 0};
   W[6, 6] = [ 0    , Wr   , 0   , 0   , 0   , 0   ;
              -Wr   , 0    , 0   , 0   , 0   , 0   ;
               0    , 0    , 0   , 0   , 0   , 0   ;
               0    , 0    , 0   , 0   , 0   , 0   ;
               0    , 0    , 0   , 0   , 0   , 0   ;
               0    , 0    , 0   , 0   , 0   , 0   ];
  R[6, 6] = diagonal({Rs, Rs, Rfd, Rkd, Rkq1, Rkq2});
end SM;
```

Figure 3. Implementation of synchronous machine model with magnetic saturation in Modelica. The saturation formulation is distinguished with the blue dashed frame.

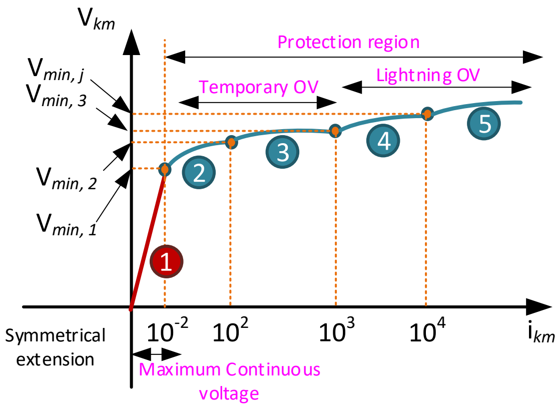


Figure 4. Voltage-current characteristic of ZnO surge arrester and operating regions.

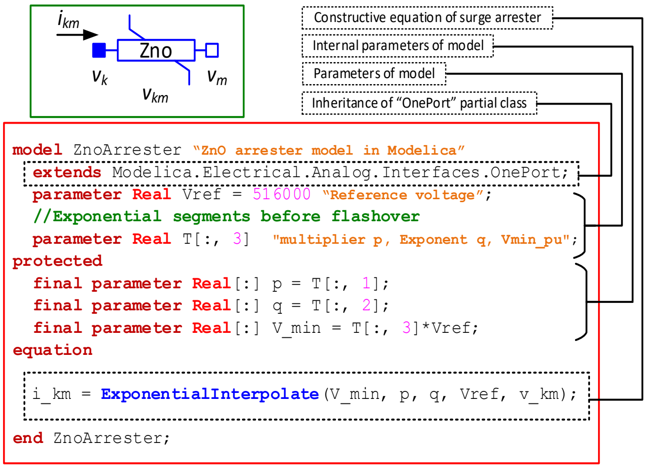
2.2 Nonlinear Arrester Modeling

Surge arresters protect the insulation of equipment, e.g., transformers in electrical systems against overvoltage transients caused by lightning or switching surges. The voltage and current characteristic of a gapless metal-oxide surge arrester as illustrated in Figure 4 is a severely nonlinear resistor with an *infinite* slope in the normal operation region and an almost *horizontal* slope in the protection region (temporary and lightning overvoltages). In EMTP, the nonlinear resistance is represented by the following *power* function:

$$i_{km} = p_j \left(\frac{v_{km}}{V_{ref}} \right)^{q_j} \tag{21}$$

where i_{km} and v_{km} are arrester current and voltage, j is the segment number starting at the voltage V_{min_j}, multiplier p_j and exponent q_j are coefficients defined for each V_{min_j} and V_{ref} is the arrester reference voltage. A *linear* function is used for the first segment.

The technique for modeling a nonlinear resistance (arrester function) is like the one used for the nonlinear



i_{km}
Zno
v_k
v_m
v_{km}
Constructive equation of surge arrester
Internal parameters of model
Parameters of model
Inheritance of “OnePort” partial class

```
model ZnoArrester “ZnO arrester model in Modelica”
  extends Modelica.Electrical.Analog.Interfaces.OnePort;
  parameter Real Vref = 516000 “Reference voltage”;
  //Exponential segments before flashover
  parameter Real T[:, 3]  "multiplier p, Exponent q, Vmin_pu";
protected
  final parameter Real[:] p = T[:, 1];
  final parameter Real[:] q = T[:, 2];
  final parameter Real[:] V_min = T[:, 3]*Vref;
equation

  i_km = ExponentialInterpolate(V_min, p, q, Vref, v_km);

end ZnoArrester;
```

Figure 5. Implementation of the ZnO surge arrester model in Modelica environment.

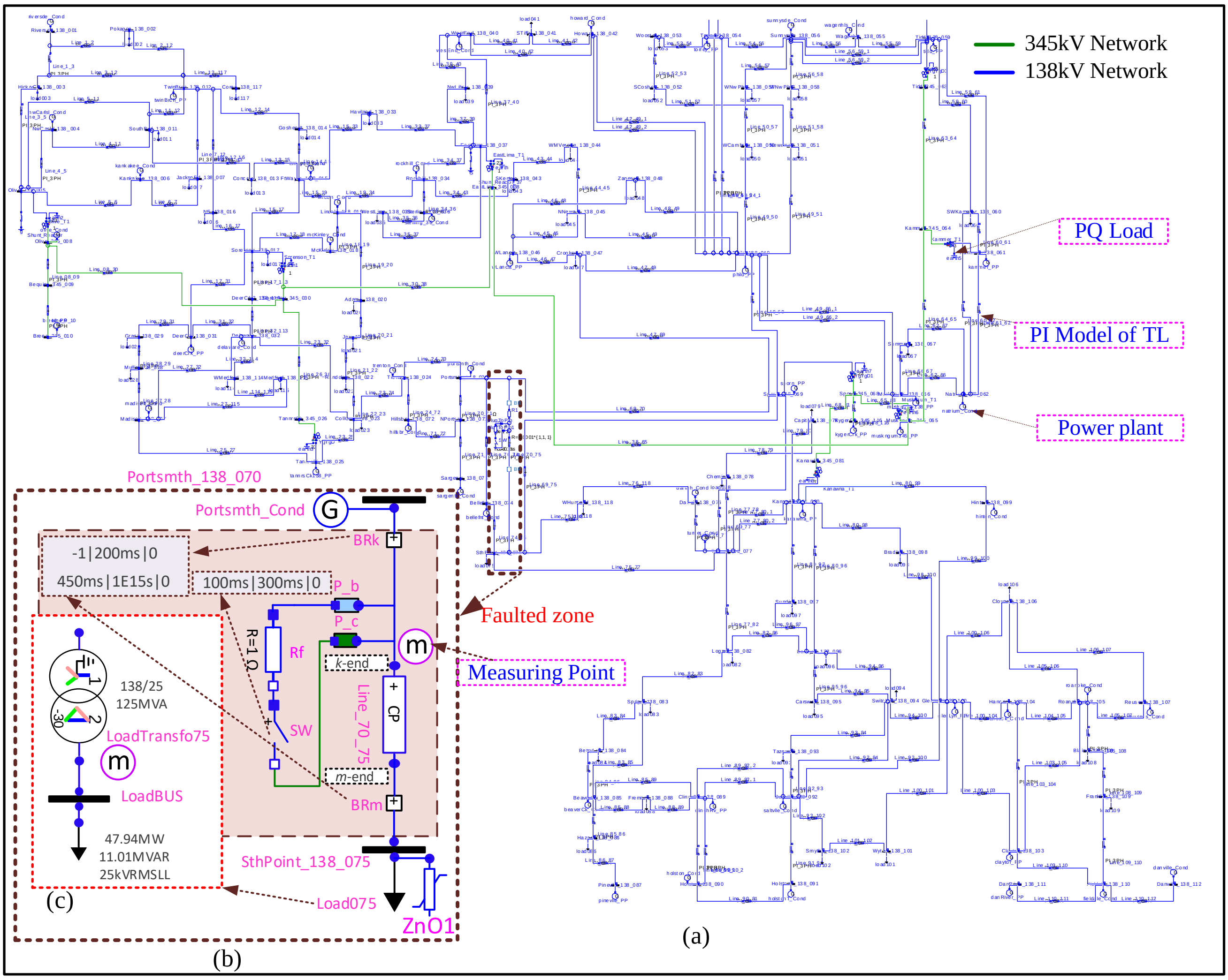


Figure 6. (a): IEEE 118-bus Network including 177 PI-section models of TL sketched using the Modelica GUI. (b): the faulty zone; a phase-*b*-to-phase-*c* fault at *k*-end of Line_70_75. The powerplant “Portsmth_Cond” is selected for validation of SM with saturation in Case 2, Surge arrester ZnO1 is inserted in the circuit only for Case 3. (c): the sub-circuit of Load75 including a saturable transformer model and constant-impedance model of load.

inductance. Figure 5 illustrates the code for the implementation of the surge arrester. The parameters of p_j, q_j and V_{min_j} are defined by n -by-3 matrix, `T`. `ExponentialInterpolate()` is a function defined by the specific class `function`, where the operating voltage is searched for the appropriate segment, j. Then, the value of i_{km} is exponentially interpolated using (21). The properties of partial class `OnePort` are inherited to apply the appropriate equations of one-port devices.

As one can see, the implementation of the model is very straightforward, there is no limitation for connection of this model in series to current sources, or inductors. Solutions converge for very small time steps in the range of nanoseconds without any numerical errors (Masoom et al. 2021).

3 Model Verification and Validation

This section presents simulation results of the modified IEEE 118-bus benchmark (Haddadi, Mahseredjian, et al. 2018) which is used to validate the accuracy of the proposed models. The same test case is also simulated with EMTP.

Figure 6.a shows the schematic diagram of the IEEE 118-bus network sketched using the developed EMT library in Modelica. A user-friendly Graphical User Interface (GUI) with an illustrative icon is designed for each component model for entering the parameters and drawing networks easily. The physical connection of components is carried out by interconnecting the terminals of appropriate components.

The IEEE-118 bus circuit consists of 54 generating units with controls (a few power plants contain more than one SM; the total number of SMs is 69), 177 transmission lines (RL coupled), 9 three-winding grid transformers, 145 two-winding transformers (91 Yd1-connected load-serving transformers+ 54 generator transformers), and 91 three-phase loads. The voltage levels are 345kV transmission, 138kV sub-transmission, 25kV distribution, and {20, 15, 10.5} kV generation. The network includes

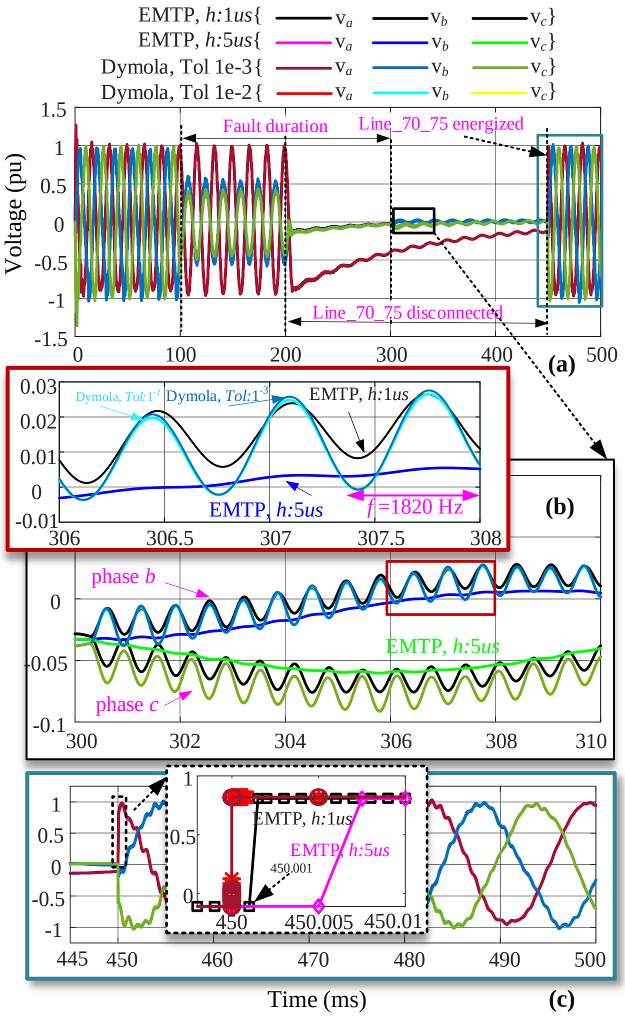


Figure 7. (a): Voltage waveforms of phases *a*, *b* and *c* at the *k*-end of Line_70_75; (b): comparison of results for the phases *b* and *c* for different solvers' parameters. (c): voltage waveforms after re-energization of Line_70_75; the close-up at the instant of closing the breakers BRk and BRm.

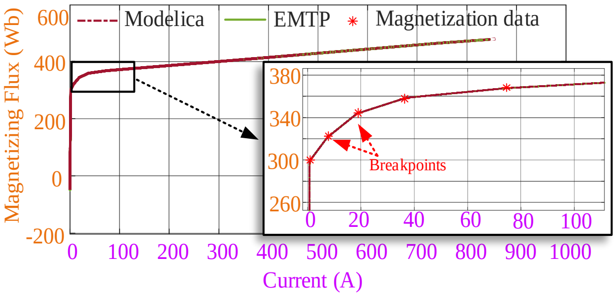


Figure 8. Current-Flux curve of magnetization branch in the LoadTransfo75 transformer; close-up of Modelica and EMTP solutions near to the knee-point.

519 nonlinear inductances and 1909 RLC elements. All SMs use a single-mass Wye-grounded model including the normalized saturation characteristics represented by 7 points. The SM control systems consist of exciter type ST1, steam turbine and governor type IEESGO, synchronous machine phasor and power system stabilizer type PSS1A. The model of all three-phase transformers consists of three single-phase units. The nonlinear magnetization branch is placed on the high-voltage side. The model uses a piecewise linear current-flux curve defined by 8 points (in the positive part of the symmetric characteristic) to represent saturation. All loads are represented by a constant impedance model.

The transmission lines (TL) are modeled using pi-sections. The constant parameter line model with propagation delay is simulated slowly, owing to the very high computational cost of the Modelica built-in delay operator. Simulation of the network starts with zero initial states.

3.1 Case 1: Phase-to-Phase Fault Analysis

For creating a transient disturbance, (see Figure 6.b), a temporary phase-to-phase fault with a fault resistance of 1 Ω is applied on the phases '*b*' and '*c*' of "Line_70_75" at *t* =100 ms followed by the isolation of the line at *t*= 200 ms (i.e. the breakers BRm and BRk open simultaneously after 6 cycles). The fault is cleared at *t* = 300 ms, then the line is reconnected at *t* = 450 ms.

Re-energizing the TL introduces high-frequency transient oscillations and allows to investigate the accuracy of transformer models in nonlinear regions.

For this purpose, the curve of flux versus current for LoadTransfo75 which is located near the faulty line is compared with EMTP as well.

Numerical tests are performed using the variable-step DASSL solver (Petzold 1982) in ODE mode with the tolerance of 1e-3 and the maximum integration order of 5 in Dymola 2021x. In EMTP, Trapezoidal/Backward Euler integrator with the step sizes of 1 μs and 5 μs is employed. The simulation time is 500 ms. The network model in Modelica contains 96308 acausal DAEs. The total number of network nodes and the size of the main system of equations in EMTP are 2533 and 3773, respectively.

Figure 7.a depicts the voltage waveforms of phases *a*, *b* and *c* at the *k*-end of Line_70_75 obtained by the two simulators with different precisions. An excellent agreement is observed between the results. Figure 7.b shows the simulation results for the phases *b* and *c* in the interval of [300, 310] ms, i.e., after the fault is removed. The results produced by Modelica models are almost identical to EMTP when step size is 1 μs (black curve), whilst the high-frequency transient oscillations (*f*=1820 Hz) are not captured by EMTP when Δt = 5 μs (blue curve). Figure 7.c depicts the curves of voltage after the re-energization of TL. The consistent results between Modelica and EMTP are observed in this period once more. The close-up view of the phase *a* voltage waveform at the instant of closing the breakers BRm and BRk shows that Modelica voltage waveform rises precisely at *t* = 450 ms while in EMTP it goes up in the next time point. The close-up illustrates the discontinuity treatment discrepancies between the two simulators. This is an

DOI 10.3384/ecp21181277

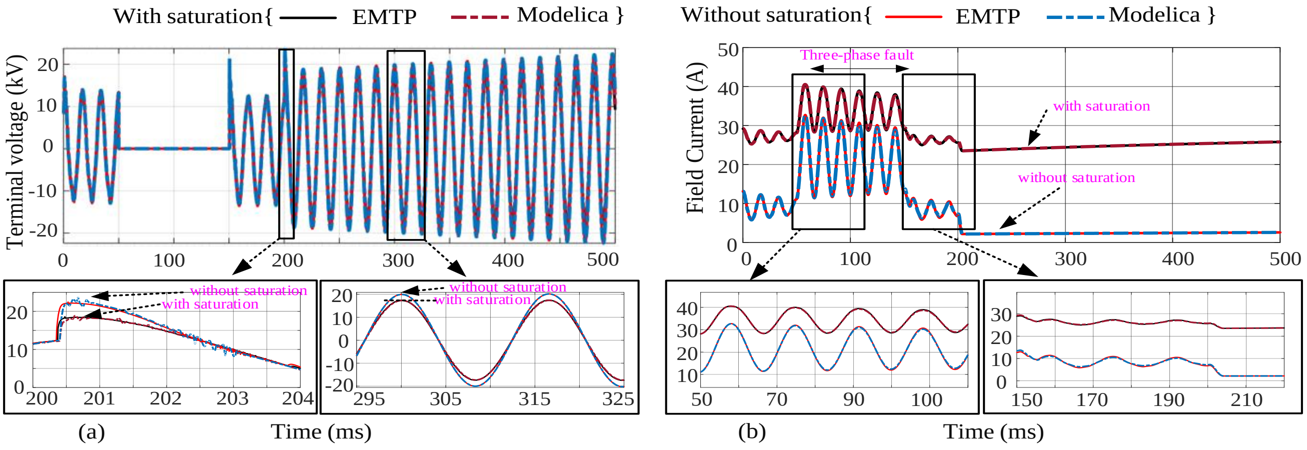


Figure 9. (a): Waveform of phase-*a* stator voltage of SM with and without saturation model; the close-up after load rejection. (b): field current with and without saturation model; the zoomed views during and after fault.

Table 1. Case 1: comparison of simulation performance.

Characteristics	Dymola		EMTP		
Solver	DASSL		Trapezoidal /Backward Euler		
Tolerance	1e-3	1e-2	Δt: 1 μs	Δt: 5 μs	Δt: 10 μs
Δt_{MIN}	0.115 fs	0.116 fs			
Δt_{MAX}	5.79 μs	0.16 μs			
No result points	203035	335261	601757	154367	81 661
No accepted steps	203034	335260	Not applicable		
f-evaluations	415437	760052	Not applicable		
J-evaluations	7393	337458	Not applicable		
CPU time (s)	371.2	1510.6	110.1	44.2	23.5
CPU-time for 1 accepted steps	1.83 ms	4.49 ms	0.18 ms	0.28 ms	0.28 ms
Performance ratio	1	0.24	3.37	8.39	15.79

important issue for the simulation of circuits with high-frequency switching.

For validating the accuracy of nonlinear components, the magnetization branch curve of LoadTransfo75 (see Figure 6.c) is examined in Figure 8. Once again, the results obtained by the two models show an excellent agreement and transformer operating points (depicted by the red dashed line) move on the transformer current-flux characteristics (distinguished by the solid red line). The *iterative* solution allows reproducing the nonlinear function accurately in both tools.

The number of nonlinear components and control closed loops has a significant impact on the accuracy and speed of simulation. For example, simulation of the same network, that is IEEE 118-bus, jams in Simscape Electrical Specialized Power Systems (SPS) package (Simscape Electrical 2020) which is comparable to Modelica environment in some ways. This package is based on the *state-space* representation of the linear network in a loop with external current sources denoting the nonlinear components. In Modelica, nonlinear functions are solved *simultaneously* through *iterative* methods which gives the most accurate results.

Table 1 shows the data and run-times of simulations carried out in Dymola and EMTP. The CPU times are extracted from the average of 5-times “re-simulations”. In Dymola, simulation is accomplished with 203034 steps in 371.2 s, which yields 1.83 ms for each time step. EMTP outperforms Dymola with the ratio of 3.37:1 when the least error is favorite, i.e., $\Delta t = 1$ µs.

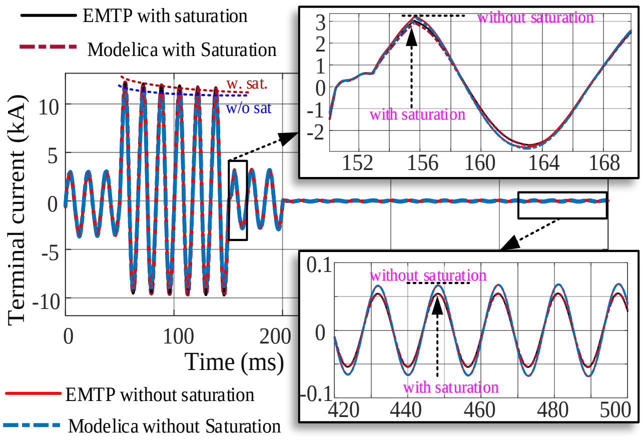


Figure 10. Phase-*a* stator current with and without saturation model; zoomed view after removing the fault and load rejection.

Tolerance has a significant impact on the CPU time and the number of time steps for the DASSL since the local error is tightly coupled with the logic for selecting the step size and order of integration. In this experiment, the simulation is repeated with the tolerance of 1e-2 as well. It causes a considerable increase in the number of time steps, Jacobian, and function evaluations. Consequently, the CPU time increases with the ratio of 4:1, whereas the accuracy of simulation does not change effectively (see Figure 7.b). The norm of error between these two simulations is reported 4.8e-3 for phase *b*. In both tolerances, the results are practically identical to EMTP when $\Delta t = 1$ µs .

However, it should be noted that the solution methods in Modelica and EMTP are fundamentally different, and a direct comparison of variable step solver with fixed-step one is not so fair. The time steps selected in Table 1 are for demonstration/comparison purposes; in reality, it is possible to select even higher time steps without significant loss of accuracy.

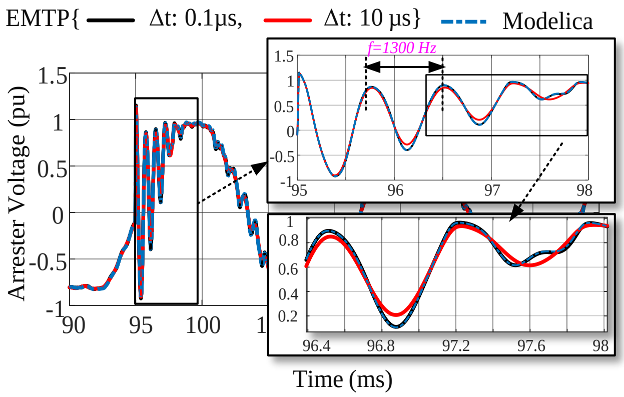


Figure 11. Voltage waveform of surge arrester ZnO1 on the bus SthPoint_138_075, DASLL solver: Tol=1e-3, EMTP solver: Trapezoidal /BE with Δt=0.1 μs and 10 μs.

3.2 Case 2: Analysis of Saturation in SM

To verify the validity of the SM model in the saturation region, a large disturbance including a sudden three-phase short-circuit fault is applied at t = 50 ms near to the terminals of SM "Portsmth_Cond" and lasts till t =150 ms. The SM protective relays detect the fault and trip the generator breaker at t = 200 ms. The parameters of both solvers are the same as Case 1, e.g., Tol=1e-3 in Dymola and $\Delta t = 5$ μs in EMTP.

Figure 9.a shows the phase-a stator voltage waveforms of the SM with and without magnetic saturation. As one can see, the results obtained from Modelica model are superimposed on EMTP ones. As time elapses, the difference between the results obtained by saturated and unsaturated models is more distinguishable on the voltage curves.

Figure 9.b depicts the field current curves of the SM with and without magnetic saturation. It is observed that the inclusion of saturation has an important impact on the excitation current needed for the generator operation.

Figure 10 illustrates the phase-a stator current graphs. As one can see, Modelica model yields precisely the same results as EMTP for both cases (with and without saturation). The stator current considering magnetic saturation is lower than without saturation. It is seen that the effect of saturation on the current in the sub-transient state is more than the transient state and the difference decreases as time elapses.

3.3 Case 3: Lightning

In this case, it is assumed that a lightning strike with the characteristics of 10kA, 8 /20 μs (see Appendix, equation (22) for the impulse source model) hits the phase-a of "Line_70_75" when the network is in steady state at t = 95 ms. The surge arresters (see Appendix, Table 3 for the parameters) are located on the bus "SthPoint_138_075", the nearest place to the high-voltage side of the transformer "LoadTransfo75" to protect it from transient overvoltages (see Figure 6.b). The simulation is run for 130 ms with the step sizes of 0.1 μs (depicted by black curve) and 10 μs (depicted by red curve) in EMTP. Other solvers parameters are like Case 1.

Figure 11 shows the phase-a voltage waveforms of the arrester ZnO1. As one can see the results obtained from Modelica arrester model are identical to EMTP when $\Delta t = 0.1\ \mu s$. A high frequency transient (1300 Hz) is created due to the strike of lightning.

Table 2 compares the performances of simulations in both tools. In Dymola, simulation is accomplished with 51513 steps in 87.2 s, which yields 1.69 ms for each time step. In this case, Dymola outperforms EMTP's best result, that is when $\Delta t = 0.1$ μs, with the ratio of 5.56:1.

This test case is designed to show the potential advantages of variable time step solvers over fixed time step ones (like EMTP). It is designed on the purpose to illustrate the fact that a very smalltime step used for the short duration of the very high transient has a penalizing effect on EMTP, but not on Modelica solver. Modelica integrator expectedly reverts to a very small time step only for a short duration. It would have been possible to apply lightning in EMTP at simulation time t = 0 s, and in which case the performance results would have been much better, nevertheless, our demonstration remains valid. A more practical example is the breaker arc model that also forces the usage of very small time steps and may be triggered at any point of time. It will effectively give an advantage to Modelica since in this case, it is required to capture longer simulation periods.

Table 2. Case 3: comparison of simulation performance.

Characteristics	Dymola	EMTP	
Solver	DASSL	Trapezoidal /BE	
Tolerance	1e-3	Δt: 0.1 μs	Δt: 10 μs
Δt_{MIN}	0.623 ps		
Δt_{MAX}	5.56 μs		
No result points	51514	1335308	21637
No accepted steps	51513	Not applicable	
f-evaluations	105576	Not applicable	
J-evaluations	1503	Not applicable	
CPU time (s)	87.2	485.6	9.9
CPU-time for 1 accepted steps	1.69 ms	0.36 ms	0.45 ms
Performance ratio	1	0.179	8.8

Conclusion

In this paper, Modelica programming language has been considered for EMT simulations due to its advantages for creating models at very high abstraction levels. In this paper, we have emphasized the modeling of synchronous machine including magnetic saturation and surge arrester. These two nonlinear models are validated by comparisons with EMTP in a large grid (IEEE 118-bus benchmark). It is shown that high-level modeling in Modelica is very accurate as compared to EMTP. However, the performance is not satisfactory, except when variable time step is used advantageously for very high-frequency transients of short duration in a long simulation interval. Nonetheless, in comparison with Simscape Electrical SPS package, the Modelica package demonstrates an excellent performance in the EMT simulation of large-scale networks composed of many nonlinearities.

The EMT-type package created by Modelica code is user-friendly, modular, easily expandable, and modifiable. It can be used for didactic purposes as well. Furthermore, the EMT-type models can be used for model exchange and co-simulation incorporating FMI.

This paper presents useful and practical information on currently available capabilities with Modelica for EMT simulation of large-scale grids. Future work will be oriented toward performance improvements and the inclusion of new models.

Acknowledgments

We would like to acknowledge the continuous support of the R&D team of RTE in this research.

Appendix

Lightning is represented by an impulse current source given by:

$$\mathrm{i}(t) = \mathrm{i}_m\left[e^{\alpha t} - e^{\beta t}\right] \qquad (22)$$

where $\mathrm{i}_m = 24.9\ [kA]$, $\alpha = -55\mathrm{k}\ [1/\mathrm{s}]$ and $\beta = -175\mathrm{k}\ [1/\mathrm{s}]$.

Table 3. Exponential segments before flashover for ZnO1.

Multiplier *p*	**Exponent *q***	$\mathbf{V}_{min}\ (\boldsymbol{pu})$
0.163113059479073E+02	0.240279296219978E+02	0.667857269772541E+00
0.134112947529269E+02	0.266219333383985E+02	0.107838745800672E+01
0.383838137212802E+02	0.200870413085749E+02	0.117458089341624E+01
0.115443146532003E+01	0.352906710089203E+02	0.125919561101624E+01
0.407093229412981E+03	0.111310570543275E+02	0.127478619617635E+01
0.256681175704043E+04	0.536270125014350E+01	0.137605475880033E+01

References

Bartolini, A., Casella, F., & Guironnet, A. (2019). “Towards pan-European power grid modelling in Modelica: Design principles and a prototype for a reference power system library”. In *13th International Modelica Conference*, vol. 157, pp. 627-636. Linkoping University Electronic Press.

Braun, W., Casella, F., & Bachmann, B. (2017). “Solving large-scale Modelica models: new approaches and experimental results using OpenModelica”. In *12 International Modelica Conference* pp. 557-563. Linkoping University Electronic Press. DOI: 10.3384/ecp17132557.

Fritzson, Peter (2014). *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. John Wiley & Sons.

Fritzson, P. et al. (2020), “The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development”. *Modeling, Identification and Control*, vol. 41, no. 4, pp. 241–295, 2020. DOI: 10.4173/mic.2020.4.1.

Guironnet, A. et al. (2018). “Towards an Open-Source Solution using Modelica for Time-Domain Simulation of Power Systems”. In: *2018 IEEE PES Innovative Smart Grid Technologies Conference* Europe (ISGT-Europe), pp. 1–6.

Haddadi, A., & Mahseredjian, J. (2018). Power system test cases for EMT-type simulation studies. *CIGRE, Paris, France, Tech. Rep. CIGRE WG C, 4*, 1-142.

Henningsson, E., Olsson, H., & Vanfretti, L. (2019). “DAE solvers for large-scale hybrid models.” In *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*, No. 157. Linköping University Electronic Press. DOI: 10.3384/ecp19157491

Kocar, I. and J. Mahseredjian (2016). “Accurate Frequency Dependent Cable Model for Electromagnetic Transients”. In: *IEEE Transactions on Power Delivery* 31.3, pp. 1281–1288. DOI: 10.1109/TPWRD.2015.2453335.

Kofman, Ernesto, Joaquín Fernández, and Denise Marzorati (2021). “Compact sparse symbolic Jacobian computation in large systems of ODEs”. In: *Applied Mathematics and Computation* 403, p. 126181. DOI: 10.1016/j.amc.2021.12618.

Karaagac, U., Mahseredjian, J., & Saad, O. (2011). “An efficient synchronous machine model for electromagnetic transients”. In *IEEE transactions on power delivery*, 26(4), 2456-2465. DOI: 10.1109/TPWRD.2011.2159249.

Karaagac, U., Mahseredjian, J. and Martinez-Velasco, J. A. (2017). “Synchronous machines,” in Chapter for Book “*Power System Transients: Parameter Determination*. Boca Raton, FL: CRC, 2009, ch. 5, pp.103–103.

Larsson, M. (2004). “ObjectStab-an educational tool for power system stability studies”. In: *IEEE Transactions on Power Systems*, vol. 19, no. 1, pp. 56-63, Feb. 2004. DOI: 10.1109/TPWRS.2003.821001

Lemaitre, C. and P. Panciatici (2014). “iTesla: Innovative tools for electrical system security within large areas”. In: *2014 IEEE PES General Meeting Conference Exposition*, pp. 1–2. DOI: 10.1109/PESGM.2014.6939447.

Mahseredjian, J., S. Dennetière, et al. (2007). “On a new approach for the simulation of transients in power systems”. In: *Electric Power Systems Research* 77.11. Selected Topics in Power System Transients - Part II, pp. 1514–1520. ISSN: 0378-7796. DOI: 10.1016/j.epsr.2006.08.027.

Mahseredjian, J., V. Dinavahi, and J. A. Martinez (2009). “Simulation Tools for Electromagnetic Transients in Power Systems: Overview and Challenges”. In: *IEEE Transactions on Power Delivery* 24.3, pp. 1657–1669. DOI: 10.1109/TPWRD.2008.2008480.

Masoom, Alireza et al. (2020). “Simulation of electromagnetic transients with Modelica, accuracy and performance assessment for transmission line models”. In: *Electric Power Systems Research* 189, p. 106799. DOI: 10.1016/j.epsr.2020.106799.

Masoom, Alireza et al. (2021). “Modelica-based Simulation of Electromagnetic Transients Using Dynaωo: Current Status and Perspectives”. In: *Electric Power Systems Research* 197, 107340. DOI: 10.1016/j.epsr.2021.107340.

Petzold, L. R. (1982). *Description of DASSL: a differential/algebraic system solver* (No. SAND-82-8637; CONF-820810-21). Sandia National Labs., Livermore, CA (USA).

Sana, A. R., Mahseredjian, J. et al. (1995). “Treatment of discontinuities in time-domain simulation of switched networks”. In *Mathematics and Computers in Simulation*, vol. 38, pp. 377-387. DOI: 10.1016/0378-4754(95)00047-2.

Simscape Electrical (2020), Specialized Power Systems User's Guide Release 2020b, Hydro-Quebec and Math Works, Sep. 2020.

Vanfretti, L., Rabuzin, T., Baudette, M., & Murad, M. (2016). “iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations”. In: *SoftwareX*, vol: 5, pp. 84-88. DOI: 10.1016/j.softx.2016.05.001.

Seismic Hybrid Testing using FMI-based Co-Simulation

Cláudio Gomes[1] Giuseppe Abbiati[2] Peter Gorm Larsen[1]

[1]Department of Electrical and Computer Engineering, Aarhus University, Denmark,
{claudio.gomes,pgl}@ece.au.dk
[1]Department of Civil and Architectural Engineering, Aarhus University, Denmark, abbiati@cae.au.dk

Abstract

Hybrid testing is an experimental technique extensively utilized in earthquake engineering to study the seismic response of structures. It requires coupling physical and numerical models in a closed feedback loop. Although this methodology is mature, a commonly accepted standard for orchestrating simulations and experiments is still missing. As a result, setting up a hybrid testing campaign still requires substantial system integration effort, which is often not affordable. In this paper, we propose the Functional Mockup Interface as a possible standard for orchestrating hybrid testing and discuss the limitations in enabling such support.

Keywords: earthquake engineering, hybrid testing, functional mockup interface, co-simulation, model exchange, master algorithm

1 Introduction

Modern civil engineering structures are designed to damage in a controlled manner when exposed to extreme seismic events. In principle, load-resisting systems are such that secondary structural elements behave as fuses that: i) cut-off loading introduced in primary structural elements; and ii) dissipate energy during seismic motion. As a result, the equivalent dynamic amplification factor of the structure is reduced and primary structural elements are preserved from damage. It comes with no surprise that understanding the dynamic response of structures in the inelastic regime is of central importance in earthquake engineering (Chopra 2012).

In order to develop construction technologies and design codes for seismic-resistant structures, a great deal of effort has been allocated to enable cost-effective experimentation in the last fifty years. Hybrid testing (HT), originally introduced in Japan, has rapidly replaced expensive shake table testing in response to this need (Nakashima 2020). HT is performed using hybrid physical-numerical models instead of purely physical models. A reference structural prototype is partitioned into a Physical Substructure (PS) and a Numerical Substructure (NS). The PS is tested in the laboratory by means of servo-controlled actuators whereas the NS is instantiated in a numerical simulation environment. A simulation algorithm solves the coupled equations of motion of the hybrid model, which is subjected to a realistic loading history, and updates the boundary conditions of PS and NS *on the fly* (Pan, Wang and Nakashima 2016). The PS is the focus of the experimental campaign since it comprises those structural components or sub-assemblies which lack of predictive numerical models. Conversely, the NS comprises all those parts that can be reliably replaced by numerical models (e.g., masses or components that experience a linear response regime).

The main advantage compared to shake table testing is that HT can be performed in *pseudodynamic* regime, that is, with an extended time scale, when the PS has a rate-independent restoring force (i.e., acts like a idealized spring). This assumption holds with a reasonable approximation for steel, concrete, and masonry structures. Noteworthy, in pseudodynamic HT, both inertia and damping forces of the PS are simulated numerically. Pseudodynamic HT enables full-scale experimentation with small hydraulic power. Typical testing time scales are $50-200$ times slower than wall-clock time. Real-time HT indicates the limit case of 1:1 time scale. If not specified, HT is assumed to be conducted in the pseudodynamic regime, which covers the vast majority of application cases. The paper of McCrum and Williams (2016) provides the most up-to-date review of the seismic HT methodology.

Seismic HT developed into a self-standing research area with relatively low cross-fertilization with other fields of civil engineering. HT methodologies developed in offshore (Sauder et al. 2016), fire (Sauca et al. 2021) and geotechnical engineering (Idinyang et al. 2019) are quite similar in form and maturity. However, for all cases, a standardized approach to coupling simulation and testing environments is still missing. As a result, setting up HT requires a lot of system integration effort, which is often not affordable.

Co-simulation, which is a technique to simulate a coupled system via combination of multiple black-box simulation units, shall provide a solution to this issue. The survey Gomes, Thule, Broman et al. (2018) gives a broad overview of co-simulation, spanning heterogeneous application domains like multi-body dynamics (Kübler and Schiehlen 2000) and large-scale circuit simulation (Lelarasmee, Ruehli and A. L. Sangiovanni-Vincentelli 1982; Newton and Alberto L. Sangiovanni-Vincentelli 1983). Specifically, simulation units, often developed and exported independently from each other in different Modelling & Simulation (M&S) tools, are coupled using an or-

DOI
10.3384/ecp21181287

chestration algorithm. The definition of simulation unit is not constrained to software artefacts, and therefore physical subsystems can be connected to virtual ones. From this perspective, HT is clearly a specific instance of co-simulation.

Contribution. To the best of our knowledge, no standard has emerged to enable seismic HT. As such, in this paper, we propose and evaluate the use of the FMI standard, version 2.1, to the co-simulation of a representative HS experimental setup, illustrated in Figure 1. We adapt the numerical algorithm proposed in Giuseppe Abbiati, La Salandra et al. (2018) to the FMI interface, with variants for Model Exchange and Co-simulation.

Structure. In order to demonstrate the use of FMI-based Co-simulation for seismic HT, an application example is described in Section 2. The specific co-simulation layout for the presented example is discussed in Section 3. Conclusions and future perspectives are given in Section 4.

2 Background and Case Study

This section provides background information on HT along with the definition of the experimental setup, which is used to demonstrate the application of FMI-based co-simulation. Such a case study aims at representing the class of structural systems typically investigated via seismic HT. Both hybrid model and simulation algorithm are accurately described.

2.1 Hybrid Model

The selected case study consists of a split-mass single-degree-of-freedom (S-DoF) system where both NS and PS are linear. Prototype structure, hybrid model and experimental setup, originally presented in Martin Hovmand, Giuseppe Abbiati and Vabbersgaard Andersen (2021), are illustrated in Figure 1. As can be appreciated, the PS consists of a cantilever beam with a tip mass whereas the NS is a S-DoF spring-mass-dashpot oscillator. An electric linear actuator controls the tip displacement of the cantilever beam, which corresponds the single DoF of the hybrid model. A force transducer measures the corresponding restoring force. The actuator is characterized by 300 mm stroke and 10 kN force capacity; the latter coincides with the force transducer admissible load. Both actuator controller and force transducer are connected to an industrial PC via EtherCAT. The industrial PC hosts a Python environment from which one can set the actuator position and read the corresponding restoring force using the control system API. The HT setup is installed at the Dynamisk LAB of Aarhus University. Since the NS of the selected hybrid model is defined by an analytical model, in our implementation, the same Python environment hosts both substructure models, simulation algorithm and the interface to the control system. In order to host more complex NSs, simulation algorithms are usually interfaced to an external simulation environment, typically based on the FE method. To this end, a number of middleware tools have

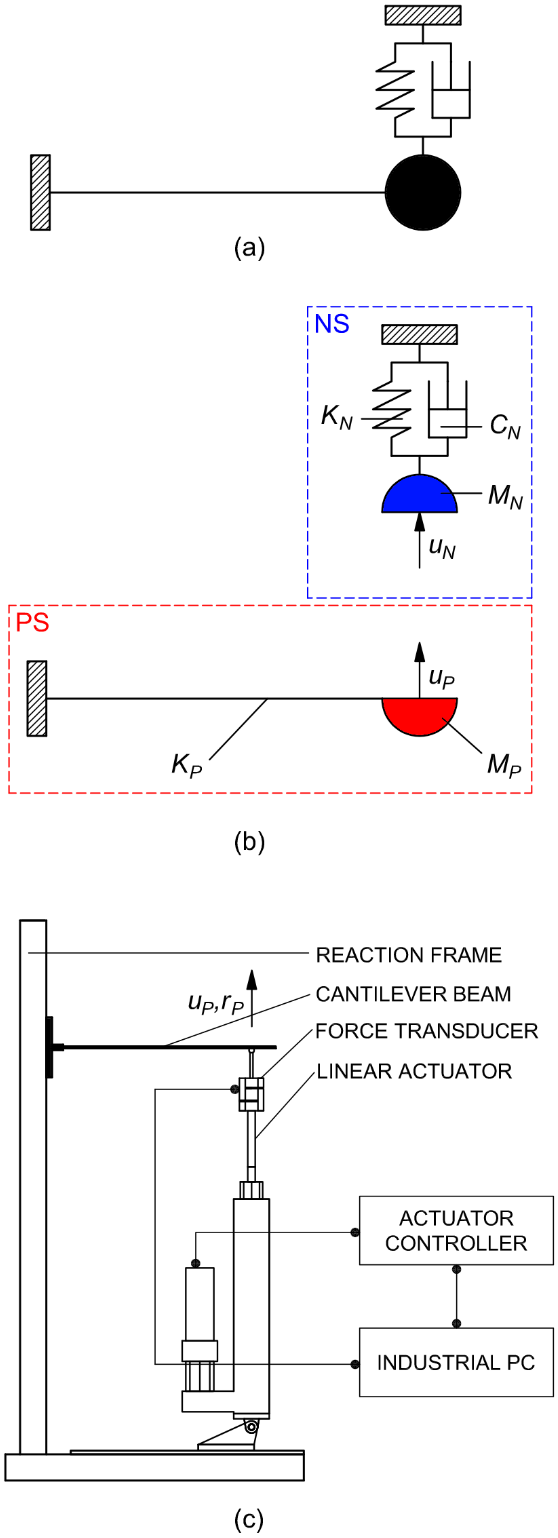


Figure 1. Reference case study adapted from M. Hovmand, G. Abbiati and Andersen (2021): a) prototype structure; b) hybrid model; c) experimental setup installed at the Dynamisk LAB of Aarhus University.

been developed. Among all, OpenFresco, developed at the University of California, Berkeley, is commonly used (McKenna, Scott and Gregory L Fenves 2010). OpenFresco can interface with industrial PCs and FE frameworks. OpenFresco is typically used in combination with OpenSees (Schellenberg, Mahin and Gregory L. Fenves 2007), which is a FE software specifically developed for analyzing steel and concrete structures subjected to earthquake loading. HT with geographically distributed PS and NS using OpenSees/OpenFresco has been demonstrated in Stojadinovic, Mosqueda and Mahin (2006). However, linking new simulation environments requires coding customized interfaces. Our contribution aims at eliminating this effort by promoting the use of FMI interface for HT.

2.2 Simulation Algorithm

In line with the philosophy of FMI-based Co-simulation, which treats the coupled simulation as a combination of independent simulation units, we selected a simulation algorithm based on partitioned time integration originally proposed in Giuseppe Abbiati, La Salandra et al. (2018) and lately adapted to hybrid fire testing in Giuseppe Abbiati, Covi et al. (2020). In a partitioned setting, Lagrange multipliers enforce compatibility conditions among substructures (i.e., simulation units) and the simulation time step is solved with a two-stage algorithm. First, substructure responses are evaluated as if they were uncoupled. Then, Lagrange multipliers are computed by solving a linearized interface problem. The original idea of using partitioned time integration to perform HT was proposed by Pegon and Magonette (2002). The first large-scale seismic testing campaign based on the scheme is described in Giuseppe Abbiati, Oreste S. Bursi et al. (2015) and Oreste S. Bursi et al. (2017).

The partitioned time integration scheme proposed in Giuseppe Abbiati, La Salandra et al. (2018) is presented to compute the seismic response of a generic hybrid model with one PS and one NS. The following system of differential-algebraic equations describes the motion of the hybrid model,

$$\begin{cases}\begin{cases}\mathbf{M}^N\ddot{\mathbf{u}}^N+\mathbf{C}^N\dot{\mathbf{u}}^N+\mathbf{r}^N\left(\mathbf{u}^N\right)=\mathbf{f}^N+\mathbf{L}^{N^T}\mathbf{\Lambda}^N\\ \mathbf{L}^N\dot{\mathbf{u}}^N+\bar{\mathbf{L}}^N\dot{\mathbf{u}}^g=\mathbf{0}\end{cases}\\ \begin{cases}\mathbf{M}^P\ddot{\mathbf{u}}^P+\mathbf{C}^P\dot{\mathbf{u}}^P+\mathbf{r}^P\left(\mathbf{u}^P\right)=\mathbf{f}^P+\mathbf{L}^{P^T}\mathbf{\Lambda}^P\\ \mathbf{L}^P\dot{\mathbf{u}}^P+\bar{\mathbf{L}}^P\dot{\mathbf{u}}^g=\mathbf{0}\end{cases}\\ \bar{\mathbf{L}}^{N^T}\mathbf{\Lambda}^N+\bar{\mathbf{L}}^{P^T}\mathbf{\Lambda}^P=\mathbf{0}\end{cases}\tag{1}$$

In detail, $\mathbf{u}^{(\bullet)}$, $\dot{\mathbf{u}}^{(\bullet)}$ and $\ddot{\mathbf{u}}^{(\bullet)}$ are displacement, velocity and acceleration vectors, respectively. The terms $\mathbf{r}^{(\bullet)}$ are rate-independent restoring force vectors. $\mathbf{M}^{(\bullet)}$ and $\mathbf{C}^{(\bullet)}$ are mass and damping matrices, respectively. Seismic loading is defined as $\mathbf{f}^{(\bullet)}=\mathbf{M}^{(\bullet)}\mathbf{t}^{(\bullet)}a(t)$ where $\mathbf{t}^{(\bullet)}$ are Boolean vectors that project seismic acceleration history $a(t)$ on system DoFs. Matrices $\mathbf{L}^{(\bullet)}$ and $\bar{\mathbf{L}}^{(\bullet)}$ localize the shared DoFs on each substructure DoF vector and the vector of generalized interface velocities $\dot{\mathbf{u}}_g$, respectively. The latter gathers all DoFs shared among substructures. The entries of $\mathbf{L}^{(\bullet)}$ are 0 and 1 whereas the entries of $\bar{\mathbf{L}}^{(\bullet)}$ are 0 and -1. According to (1), Lagrange multiplier vectors $\mathbf{\Lambda}^{(\bullet)}$ enforce velocity compatibility with $\dot{\mathbf{u}}_g$. It is important to stress that all Lagrange multiplier vectors form a set of self-balanced forces to ensure interface equilibrium *a priori*. The solution of (1) enforces kinematic compatibility *a posteriori*. The HT algorithm is presented to integrate (1) from time t_k to t_{k+1} with a time step Δt.

1. Solve the NS *free* problem at t_{k+1},

$$\begin{cases}\tilde{\mathbf{u}}_{k+1}^{N,free}=\mathbf{u}_k^N+\dot{\mathbf{u}}_k^N\Delta t+\left(\frac{1}{2}-\beta^N\right)\Delta t^2\ddot{\mathbf{u}}_k^N\\ \tilde{\dot{\mathbf{u}}}_{k+1}^{N,free}=\dot{\mathbf{u}}_k^N+\left(1-\gamma^N\right)\Delta t\ddot{\mathbf{u}}_k^N\end{cases}\tag{2}$$

$$\ddot{\mathbf{u}}_{k+1}^{N,free}=\mathbf{D}^{N^{-1}}\left[\mathbf{f}_{k+1}^N-\mathbf{C}^N\tilde{\dot{\mathbf{u}}}_{k+1}^{N,free}-\mathbf{r}_{k+1}^N\left(\tilde{\mathbf{u}}_{k+1}^{N,free}\right)\right]\tag{3}$$

$$\begin{cases}\mathbf{u}_{k+1}^{N,free}=\tilde{\mathbf{u}}_{k+1}^{N,free}+\ddot{\mathbf{u}}_{k+1}^{N,free}\beta^N\Delta t^2\\ \dot{\mathbf{u}}_{k+1}^{N,free}=\tilde{\dot{\mathbf{u}}}_{k+1}^{N,free}+\ddot{\mathbf{u}}_{k+1}^{N,free}\gamma^N\Delta t\end{cases}\tag{4}$$

with,

$$\mathbf{D}^N=\mathbf{M}^N+\mathbf{C}^N\gamma^N\Delta t+\mathbf{K}^N\beta^N\Delta t^2\tag{5}$$

where γ^N and β^N are the parameters of the Newmark scheme for the NS (Bathe 1982) and $\mathbf{K}^N$ is the stiffness matrix. In Equation (3), the displacement predictor $\tilde{\mathbf{u}}_{k+1}^{N,free}$ is sent to an external FE software that computes the corresponding restoring force $\mathbf{r}_{k+1}^N$.

2. Solve the PS *free* problem at t_{k+1},

$$\begin{cases}\tilde{\mathbf{u}}_{k+1}^{P,free}=\mathbf{u}_k^P+\dot{\mathbf{u}}_k^P\Delta t+\left(\frac{1}{2}-\beta^P\right)\Delta t^2\ddot{\mathbf{u}}_k^P\\ \tilde{\dot{\mathbf{u}}}_{k+1}^{P,free}=\dot{\mathbf{u}}_k^P+\left(1-\gamma^P\right)\Delta t\ddot{\mathbf{u}}_k^P\end{cases}\tag{6}$$

$$\ddot{\mathbf{u}}_{k+1}^{P,free}=\mathbf{D}^{P^{-1}}\left[\mathbf{f}_{k+1}^P-\mathbf{C}^P\tilde{\dot{\mathbf{u}}}_{k+1}^{P,free}-\mathbf{r}_{k+1}^P\left(\tilde{\mathbf{u}}_{k+1}^{P,free}\right)\right]\tag{7}$$

$$\begin{cases}\mathbf{u}_{k+1}^{P,free}=\tilde{\mathbf{u}}_{k+1}^{P,free}+\ddot{\mathbf{u}}_{k+1}^{P,free}\beta^P\Delta t^2\\ \dot{\mathbf{u}}_{k+1}^{P,free}=\tilde{\dot{\mathbf{u}}}_{k+1}^{P,free}+\ddot{\mathbf{u}}_{k+1}^{P,free}\gamma^P\Delta t\end{cases}\tag{8}$$

with,

$$\mathbf{D}^P=\mathbf{M}^P+\mathbf{C}^P\gamma^P\Delta t+\mathbf{K}^P\beta^P\Delta t^2\tag{9}$$

where γ^P and β^P are the parameters of the Newmark scheme for the PS (Bathe 1982). In Equation (7), the displacement predictor $\tilde{\mathbf{u}}_{k+1}^{P,free}$ is imposed to the PS by means of servo-controlled actuators and the corresponding restoring force vector $\mathbf{r}_{k+1}^P$ is measured using force transducers. Noteworthy displacement control errors affects the measured restoring force $\mathbf{r}^{P,free}_{k+1,mes}$ and may bias the emulated system response. Accordingly, control and measurement errors are compensated as suggested by Oreste S Bursi O. S. and Shing (1996),

$$\mathbf{r}^{P,free}_{k+1} = \mathbf{r}^{P,free}_{k+1,mes} + \mathbf{K}^{P}(\mathbf{u}^{P,free}_{k+1} - \mathbf{u}^{P,free}_{k+1,mes}) \tag{10}$$

where $\mathbf{u}^{P,free}_{k+1,mes}$ and $\mathbf{r}^{P,free}_{k+1,mes}$ are measured displacement and restoring force vectors. The stiffness matrix of the PS $\mathbf{K}^P$ is estimated before HT based on low-amplitude cyclic tests.

3. Solve the linearized interface problem a t_{k+1},

$$\begin{bmatrix} \mathbf{\Lambda}^{N}_{k+1} \\ \mathbf{\Lambda}^{P}_{k+1} \\ \dot{\mathbf{u}}^{g}_{k+1} \end{bmatrix} = -\mathbf{G}^{-1} \begin{bmatrix} \mathbf{L}^{N}\dot{\mathbf{u}}^{N,free}_{k+1} \\ \mathbf{L}^{P}\dot{\mathbf{u}}^{P,free}_{k+1} \\ \mathbf{0} \end{bmatrix} \tag{11}$$

with,

$$\mathbf{G} = \begin{bmatrix} \mathbf{L}^{N}\mathbf{D}^{N^{-1}}\mathbf{L}^{N^{T}}\gamma^{N}\Delta t & \mathbf{0} & \bar{\mathbf{L}}^{N} \\ \mathbf{0} & \mathbf{L}^{P}\mathbf{D}^{P^{-1}}\mathbf{L}^{P^{T}}\gamma^{P}\Delta t & \bar{\mathbf{L}}^{P} \\ \bar{\mathbf{L}}^{N^{T}} & \bar{\mathbf{L}}^{P^{T}} & \mathbf{0} \end{bmatrix} \tag{12}$$

4. Calculate *link* kinematic quantities at t_{k+1},

$$\begin{cases} \ddot{\mathbf{u}}^{N,link}_{k+1} = \mathbf{D}^{N^{-1}}\mathbf{L}^{N^{T}}\mathbf{\Lambda}^{N}_{k+1} \\ \dot{\mathbf{u}}^{N,link}_{k+1} = \mathbf{D}^{N^{-1}}\mathbf{L}^{N^{T}}\mathbf{\Lambda}^{N}_{k+1}\gamma^{N}\Delta t \\ \mathbf{u}^{N,link}_{k+1} = \mathbf{D}^{N^{-1}}\mathbf{L}^{N^{T}}\mathbf{\Lambda}^{N}_{k+1}\beta^{N}\Delta t^{2} \end{cases} \tag{13}$$

$$\begin{cases} \ddot{\mathbf{u}}^{P,link}_{k+1} = \mathbf{D}^{P^{-1}}\mathbf{L}^{P^{T}}\mathbf{\Lambda}^{P}_{k+1} \\ \dot{\mathbf{u}}^{P,link}_{k+1} = \mathbf{D}^{P^{-1}}\mathbf{L}^{P^{T}}\mathbf{\Lambda}^{P}_{k+1}\gamma^{P}\Delta t \\ \mathbf{u}^{P,link}_{k+1} = \mathbf{D}^{P^{-1}}\mathbf{L}^{P^{T}}\mathbf{\Lambda}^{P}_{k+1}\beta^{P}\Delta t^{2} \end{cases} \tag{14}$$

5. Calculate *coupled* kinematic quantities at t_{k+1},

$$\begin{cases} \ddot{\mathbf{u}}^{N}_{k+1} = \ddot{\mathbf{u}}^{N,free}_{k+1} + \ddot{\mathbf{u}}^{N,link}_{k+1} \\ \dot{\mathbf{u}}^{N}_{k+1} = \dot{\mathbf{u}}^{N,free}_{k+1} + \dot{\mathbf{u}}^{N,link}_{k+1} \\ \mathbf{u}^{N}_{k+1} = \mathbf{u}^{N,free}_{k+1} + \mathbf{u}^{N,link}_{k+1} \end{cases} \tag{15}$$

$$\begin{cases} \ddot{\mathbf{u}}^{P}_{k+1} = \ddot{\mathbf{u}}^{P,free}_{k+1} + \ddot{\mathbf{u}}^{P,link}_{k+1} \\ \dot{\mathbf{u}}^{P}_{k+1} = \dot{\mathbf{u}}^{P,free}_{k+1} + \dot{\mathbf{u}}^{P,link}_{k+1} \\ \mathbf{u}^{P}_{k+1} = \mathbf{u}^{P,free}_{k+1} + \mathbf{u}^{P,link}_{k+1} \end{cases} \tag{16}$$

The main advantage of partitioned time integration is that the PS *free* response can be solved with an *explicit* scheme, which does not require estimating the PS stiffness $\mathbf{K}^P$, while the NS *free* response is solved with an *implicit* scheme. Compatibility of interface velocities stated in (1) ensures that the coupled simulation is stable as long as each time integration scheme is stable as uncoupled (Gravouil and Combescure 2001). Accordingly, in our implementation, the central difference scheme is utilized on the PS ($\gamma^P = \frac{1}{2}, \beta^P = 0$) whereas the mid-point rule on the NS ($\gamma^N = \frac{1}{2}, \beta^N = \frac{1}{4}$) (Bathe 1982). Usually, the PS is characterized by very few DoFs ($1-10$) and relatively low eigenfrequencies ($0-20$ Hz) so, typically, $\Delta t = 10$ msec. Considering a time scale $\lambda = 100$, the wall-clock time required to solve one time integration step is $\Delta T = \Delta t \lambda = 1$ sec, which is sufficiently large to accommodate actuation and filtering delays. In our example, $\mathbf{M}^N$ and $\mathbf{K}^N$ have constant scalar values and the NS *free* response is solved without Newton-Raphson iterations, following the operator-splitting approach (Oreste S Bursi O. S. and Shing 1996). Since, HT is performed with an extended time scale, $\mathbf{M}^P$ is estimated analytically. Following the procedure suggested in Molina et al. (2011), we set the PS damping matrix $\mathbf{C}^P$ to zero.

We stress that the Steklov-Poincaré operator $\mathbf{G}$ defined in (12) is computed and inverted once before HT. As a result, the solution of the interface problem (Step 3) and calculation of *link* kinematic quantities (Step 4) require only of a few matrix multiplications.

3 FMI-based Implementation

The FMI 2.1 defines two main interfaces: the Co-simulation (CS), and the Model Exchange (ME). In the FMI nomenclature, a simulation unit is called the Functional Mockup Unit (FMU), and it may implement one or two of the main interfaces. The FMU is a zip file containing: binaries and source code (optional) implementing the API functions; miscellaneous resources; and an XML file, describing the variables, model structure, and other data.

In this work, HT is numerically simulated. Therefore, both NS and PS FMUs are implemented as ODEs. In this regard, the difference between an the ME and the CS interfaces lies in the fact that the former enables the FMU to expose the ODE derivative function, whereas the CS always represents the time discretized sub-model, enabling the FMU to export the sequence of ODE states, as the simulation progresses in time. From the point of view of the orchestration algorithm, defined below, a FMU ME still needs to be coupled to a numerical solver, which will be responsible for querying the derivatives and updating the states of the FMU, whereas a FMU CS comes with its own numerical solver. The differences are illustrated in Figures 2 and 3.

A simulation scenario is a description of the FMUs and their inter-connections. In order to make the following discussion clearer, we define the following:

Importer – Denotes the application that loads the FMUs;

Orchestrator – Denotes the algorithm, executing on the Importer, that interacts with the FMUs, inspecting their outputs, setting their inputs, etc, according to the simulation scenario.

Coupling – Denotes the strategy that the orchestrator uses, in order to ensure that physical laws are respected.

While it is common to mix the Orchestrator and Coupling together as one concept, as we will show later, the

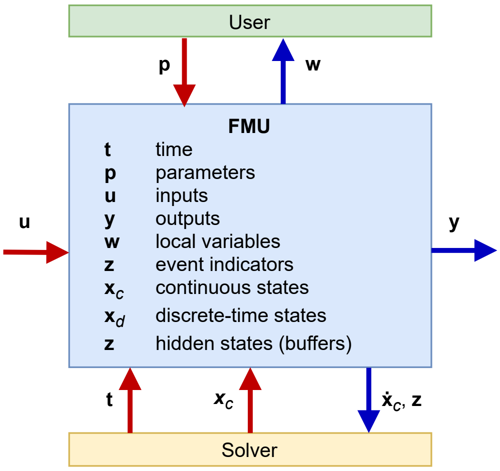


Figure 2. Schematic view of data flow between user, the solver of the importer and the FMU for Model Exchange. Based on (FMI v. 3.0 2021).

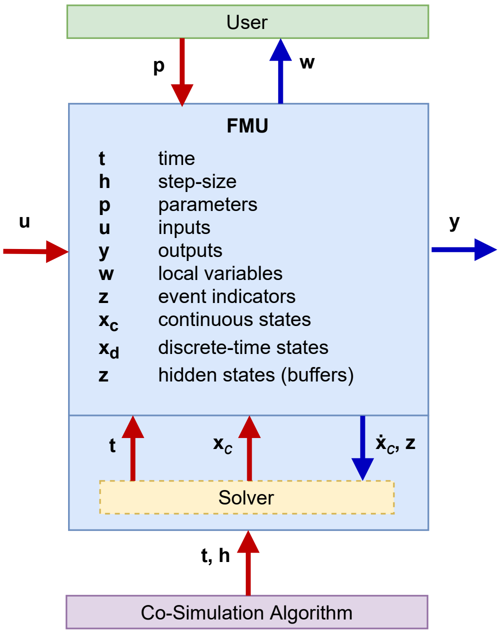


Figure 3. Schematic view of data flow between user, the solver of the importer and the FMU for Co-Simulation. Based on (FMI v. 3.0 2021).

distinction is useful because the Orchestrator is generic and can be applied to many different simulation scenarios (e.g., consider the Jacobi and Gauss-Seidel schemes (Kübler and Schiehlen 2000; Bastian et al. 2011)), but the Coupling is often specifically designed for a particular simulation scenario (e.g., the coupling can be implemented as a semantic adaptation (Gomes, Meyers et al. 2018), or automatically generated from hints (Gomes, Oakes et al. 2019; Oakes et al. 2021)).

3.1 FMI Co-simulation Interface

Figure 4 shows two possible simulation scenarios that implement the setup described in Figure 1 in a co-simulation with the coupling algorithm introduced in Section 2.2. In the figure, variants A and B differ only in the data that is communicated between the Coupling and the FMUs. In variant A, vectors $\dot{\mathbf{u}}^{\mathbf{N}}$ and $\dot{\mathbf{u}}^{\mathbf{P}}$ are copied, whereas in variant B, the (typically) smaller vectors $\mathbf{L}^{N}\dot{\mathbf{u}}^{\mathbf{N}}$ and $\mathbf{L}^{P}\dot{\mathbf{u}}^{\mathbf{P}}$ are passed. Each variant requires therefore, a slightly different FMU implementation. In the figure, variant B only shows the differences with respect to variant A for simplicity.

In both variants, the *NSFMU* and *PSFMU* communicate with the FE software and Test Setup, respectively. These FMUs have no input-to-output algebraic dependencies, and all of their computations can happen inside the `fmi2DoStep` function, making them well suited to be implemented as co-simulation FMUs.

It is important to remark that the application of the FMI CS interface implies that the FMUs are compatible with the coupling algorithm. In particular, recall that the coupling algorithm requires a correction to the states of the FMUs, in Equations (15) and (16). As such, the FMUs must support a mechanism to perform this correction. The authors of (Brembeck et al. 2014) also report on the same difficulties. The implementation of *Coupling* can too be done in multiple ways, described next.

Coupling as Custom Orchestrator. The most common approach is to implement, or automatically generate, the coupling algorithm directly as part of the orchestration algorithm. This works well when the user has full control over the orchestration algorithm, or when the co-simulation framework, or modelling environment, being used, provides the ability for customization (e.g., using a co-simulation library like PyFMI (Andersson, Åkesson and Führer 2016), FMPy[1], among others).

Coupling as FMU. For situations where the user has little programming expertize, or no control over the co-simulation framework, it is easier to implement the coupling as an FMU, which is then loaded during the co-simulation, as any other FMU. Frameworks such as the one described in (Gomes, Meyers et al. 2018), where the user is offered a domain specific language to describe common algebraic expressions, which is then used to automatically generate a new FMU that wraps the old ones

[1] https://github.com/CATIA-Systems/FMPy

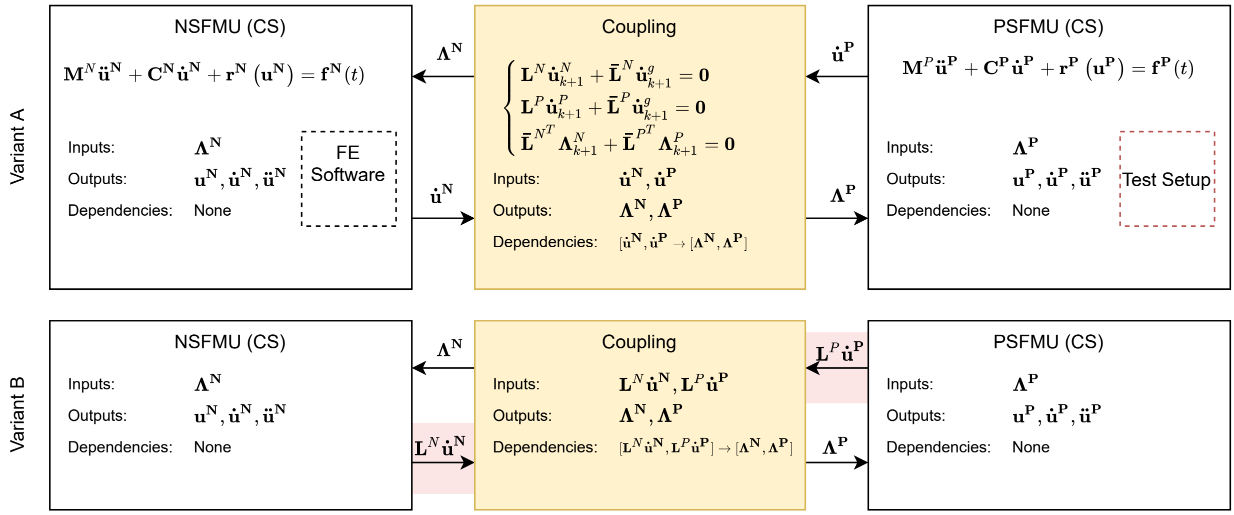

Variant A
NSFMU (CS)
$\mathbf{M}^N\ddot{\mathbf{u}}^\mathbf{N}+\mathbf{C}^\mathbf{N}\dot{\mathbf{u}}^\mathbf{N}+\mathbf{r}^\mathbf{N}\left(\mathbf{u}^\mathbf{N}\right)=\mathbf{f}^\mathbf{N}(t)$
Inputs: $\mathbf{\Lambda}^\mathbf{N}$
Outputs: $\mathbf{u}^\mathbf{N},\dot{\mathbf{u}}^\mathbf{N},\ddot{\mathbf{u}}^\mathbf{N}$
Dependencies: None
FE Software
$\mathbf{\Lambda}^\mathbf{N}$
$\dot{\mathbf{u}}^\mathbf{N}$
Coupling
$\begin{cases}\mathbf{L}^N\dot{\mathbf{u}}^N_{k+1}+\bar{\mathbf{L}}^N\dot{\mathbf{u}}^g_{k+1}=\mathbf{0}\\ \mathbf{L}^P\dot{\mathbf{u}}^P_{k+1}+\bar{\mathbf{L}}^P\dot{\mathbf{u}}^g_{k+1}=\mathbf{0}\\ \bar{\mathbf{L}}^{NT}\mathbf{\Lambda}^N_{k+1}+\bar{\mathbf{L}}^{PT}\mathbf{\Lambda}^P_{k+1}=\mathbf{0}\end{cases}$
Inputs: $\dot{\mathbf{u}}^\mathbf{N},\dot{\mathbf{u}}^\mathbf{P}$
Outputs: $\mathbf{\Lambda}^\mathbf{N},\mathbf{\Lambda}^\mathbf{P}$
Dependencies: $[\dot{\mathbf{u}}^\mathbf{N},\dot{\mathbf{u}}^\mathbf{P}\to[\mathbf{\Lambda}^\mathbf{N},\mathbf{\Lambda}^\mathbf{P}]$
$\dot{\mathbf{u}}^\mathbf{P}$
$\mathbf{\Lambda}^\mathbf{P}$
PSFMU (CS)
$\mathbf{M}^P\ddot{\mathbf{u}}^\mathbf{P}+\mathbf{C}^\mathbf{P}\dot{\mathbf{u}}^\mathbf{P}+\mathbf{r}^\mathbf{P}\left(\mathbf{u}^\mathbf{P}\right)=\mathbf{f}^\mathbf{P}(t)$
Inputs: $\mathbf{\Lambda}^\mathbf{P}$
Outputs: $\mathbf{u}^\mathbf{P},\dot{\mathbf{u}}^\mathbf{P},\ddot{\mathbf{u}}^\mathbf{P}$
Dependencies: None
Test Setup
Variant B
NSFMU (CS)
Inputs: $\mathbf{\Lambda}^\mathbf{N}$
Outputs: $\mathbf{u}^\mathbf{N},\dot{\mathbf{u}}^\mathbf{N},\ddot{\mathbf{u}}^\mathbf{N}$
Dependencies: None
$\mathbf{\Lambda}^\mathbf{N}$
$\mathbf{L}^N\dot{\mathbf{u}}^\mathbf{N}$
Coupling
Inputs: $\mathbf{L}^N\dot{\mathbf{u}}^\mathbf{N},\mathbf{L}^P\dot{\mathbf{u}}^\mathbf{P}$
Outputs: $\mathbf{\Lambda}^\mathbf{N},\mathbf{\Lambda}^\mathbf{P}$
Dependencies: $[\mathbf{L}^N\dot{\mathbf{u}}^\mathbf{N},\mathbf{L}^P\dot{\mathbf{u}}^\mathbf{P}]\to[\mathbf{\Lambda}^\mathbf{N},\mathbf{\Lambda}^\mathbf{P}]$
$\mathbf{L}^P\dot{\mathbf{u}}^\mathbf{P}$
$\mathbf{\Lambda}^\mathbf{P}$
PSFMU (CS)
Inputs: $\mathbf{\Lambda}^\mathbf{P}$
Outputs: $\mathbf{u}^\mathbf{P},\dot{\mathbf{u}}^\mathbf{P},\ddot{\mathbf{u}}^\mathbf{P}$
Dependencies: None

Figure 4. Illustration of two possible simulation scenarios that implement the setup described in Figure 1 in a co-simulation with the coupling algorithm introduced in Section 2.2. The dashed boxes represent the fact that NSFMU (respectively PSFMU) communicate with a FE Software (resp. the Test Setup), when the `fmi2DoStep` function is invoked. Variant B only shows the differences (highlighted in red) with respect to variant A. Each variant requires a different FMU implementation, with variant B allowing for less communication.

with those extra expressions, can be used for this. In addition, there are libraries that facilitate the implementation of FMUs, such as Moka (Aslan, Durak and Taylan 2015) and PyFMU (Legaard et al. 2020) (the later allows the user to program an FMU in Python).

However, for the scenario described in Figure 4, special attention must be payed to the algebraic dependencies: since FMI 2.1 supports no feedthrough, the computation of *Coupling* must be carried out in the `fmi2DoStep` function. This means that the `fmi2DoStep` of the *Coupling* FMU must be invoked after having provided the new inputs, originated from the outputs of the already stepped *NSFMU* and *PSFMU*. The consequence is that the co-simulation framework must accept some kind of description of the ordering in which the FMUs shall be stepped. The FMI standard 2.1 offers no mechanism to provide this information, even though it plays an important role in reducing co-simulation errors (Gomes, Thule, Lúcio et al. 2020; Oakes et al. 2021; Gomes, Oakes et al. 2019; Gomes, Lucio and Vangheluwe 2019).

Consistent Initialization. The implemented coupling algorithm, needs to be initialized with the jacobian of *NSFMU* and *PSFMU*. Since we had control over the implementation of these FMUs, we choose to simply expose these as outputs. In other cases, the `fmi2GetDirectionalDerivative` can be used to obtain this information.

3.2 FMI Model Exchange Interface

When implementing the scenarios described in Figure 4 using the FMI ME interface, the main difference lies in the ability to easily perform the state corrections described by Equations (15) and (16). This is illustrated in Figure 5.

The advantage is that the FMU is independent of the coupling algorithm, making this approach ideal for when the users have no control over the implementation of the FMUs (e.g., as in the case of FMI export features in M&S tools). The downside is the added complexity of the orchestration algorithm and the FMU exporter, which must also implement a greater number of FMI functions (compared to FMI CS) and ensure that the internal state is organized in a single vector.

4 Results and Conclusion

This paper describes multiple ways in which the FMI 2.1 for Model Exchange (ME) and Co-Simulation (CS) interfaces, can be used for the implementation of hybrid testing. To the best of our knowledge, there has not yet been any application of the FMI to the field of seismic hybrid testing and no standard has emerged to enable this. Having such standard would greatly facilitate the coupling of heterogeneous codes and facilitate the exchange of numerical models.

The approaches are discussed within the context of their implementation in an hybrid testing setup, installed at the Dynamisk LAB of Aarhus University. A snapshot of the numerical results can be seen in Figure 6, and a video recording of the experiment is available online at `https://youtu.be/-VkrQJaUo1o`.

The conclusion is that the both FMI ME and CS interfaces are well suited for this task, provided that the co-simulation framework supports minor customization of the ordering in which the FMUs are stepped in simulated time.

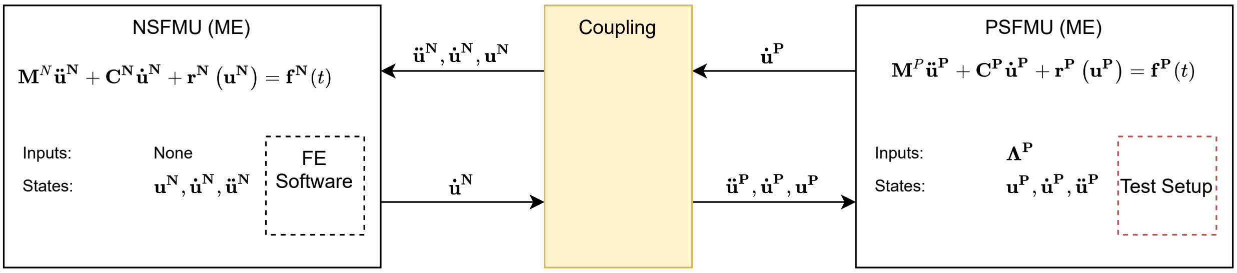

NSFMU (ME)
$\mathbf{M}^N\ddot{\mathbf{u}}^N + \mathbf{C}^N\dot{\mathbf{u}}^N + \mathbf{r}^N(\mathbf{u}^N) = \mathbf{f}^N(t)$
Inputs:
None
States:
$\mathbf{u}^N, \dot{\mathbf{u}}^N, \ddot{\mathbf{u}}^N$
FE Software
$\ddot{\mathbf{u}}^N, \dot{\mathbf{u}}^N, \mathbf{u}^N$
$\dot{\mathbf{u}}^N$
Coupling
$\dot{\mathbf{u}}^P$
$\ddot{\mathbf{u}}^P, \dot{\mathbf{u}}^P, \mathbf{u}^P$
PSFMU (ME)
$\mathbf{M}^P\ddot{\mathbf{u}}^P + \mathbf{C}^P\dot{\mathbf{u}}^P + \mathbf{r}^P(\mathbf{u}^P) = \mathbf{f}^P(t)$
Inputs:
$\mathbf{\Lambda}^P$
States:
$\mathbf{u}^P, \dot{\mathbf{u}}^P, \ddot{\mathbf{u}}^P$
Test Setup

Figure 5. Illustration of simulation scenario implementing the setup of Figure 1 using the FMI Model Exchange interface. The coupling implementation is the same as in Figure 4. The dashed boxes represent the fact that NSFMU (respectively PSFMU) communicate with a FE Software (resp. the Test Setup), when the `fmi2GetReal` function is invoked.

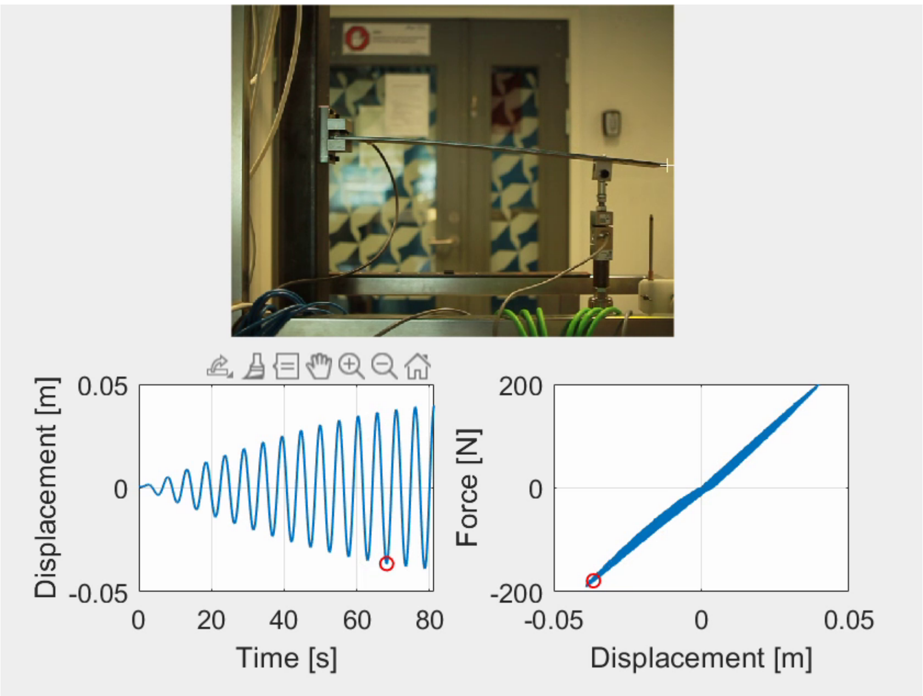


Figure 6. Numerical results using the FMI Co-simulation Interface. The results with Model Exchange are similar. The full video can be seen online at `https://youtu.be/-VkrQJaUolo`.

Acknowledgements

We are grateful for the support from the Poul Due Jensen Foundation of the Centre for Digital Twin Technology at Aarhus University.

References

Abbiati, Giuseppe, Oreste S. Bursi et al. (2015-10). "Hybrid simulation of a multi-span RC viaduct with plain bars and sliding bearings". en. In: *Earthquake Engineering & Structural Dynamics* 44.13. ZSCC: NoCitationData[s0], pp. 2221–2240. ISSN: 00988847. DOI: 10.1002/eqe.2580. URL: http://doi.wiley.com/10.1002/eqe.2580 (visited on 2020-03-04).

Abbiati, Giuseppe, Patrick Covi et al. (2020-09). "A Real-Time Hybrid Fire Simulation Method Based on Dynamic Relaxation and Partitioned Time Integration". en. In: *Journal of Engineering Mechanics* 146.9, p. 04020104. ISSN: 0733-9399, 1943-7889. DOI: 10.1061/(ASCE)EM.1943-7889.0001826.

Abbiati, Giuseppe, Vincenzo La Salandra et al. (2018-02). "A composite experimental dynamic substructuring method based on partitioned algorithms and localized Lagrange multipliers". en. In: *Mechanical Systems and Signal Processing* 100, pp. 85–112. ISSN: 08883270. DOI: 10.1016/j.ymssp.2017.07.020. URL: https://linkinghub.elsevier.com/retrieve/pii/S0888327017303849 (visited on 2020-03-05).

Andersson, Christian, Johan Åkesson and Claus Führer (2016). *Pyfmi: A Python Package for Simulation of Coupled Dynamic Models with the Functional Mock-up Interface*. Centre for Mathematical Sciences, Lund University Lund.

Aslan, Memduha, Umut Durak and Koray Taylan (2015-07). "MOKA: An Object-Oriented Framework for FMI Co-Simulation". In: *Conference on Summer Computer Simulation*. Chicago, Illinois: Society for Computer Simulation International San Diego, CA, USA, pp. 1–8.

Bastian, Jens et al. (2011-06). "Master for Co-Simulation Using FMI". In: *8th International Modelica Conference*. Dresden, Germany: Linköping University Electronic Press, Linköpings universitet, pp. 115–120. DOI: 10.3384/ecp11063115.

Bathe, Klaus-JOrgen (1982). "Finite Element Procedures for Solids and Structures LinearAnalysis". In: *Finite Element Procedures*, pp. 148–214.

Brembeck, Jonathan et al. (2014-03). "Nonlinear State Estimation with an Extended FMI 2.0 Co-Simulation Interface". In: *10th International Modelica Conference*. Lund, Sweden: Linköping University Electronic Press; Linköpings universitet, pp. 53–62. DOI: 10.3384/ecp1409653.

Bursi O. S., Oreste S and Pui-Shum B. Shing (1996). "Evaluation of Some Implicit Time-Stepping Algorithms for Pseudodynamic Tests". en. In: *Earthquake Engineering & Structural Dynamics* 25, pp. 333–355.

Bursi, Oreste S. et al. (2017-11). "Nonlinear heterogeneous dynamic substructuring and partitioned FETI time integration for the development of low-discrepancy simulation models". en. In: *International Journal for Numerical Methods in Engineering* 112.9, pp. 1253–1291. ISSN: 00295981. DOI: 10.1002/nme.5556. URL: http://doi.wiley.com/10.1002/nme.5556 (visited on 2020-03-04).

Chopra, Anil K. (2012). *Dynamics of structures: theory and applications to earthquake engineering*. en. 4th ed. ZSCC: 0000002. Upper Saddle River, N.J: Prentice Hall. ISBN: 978-0-13-285803-8.

FMI v. 3.0 (2021). *Functional Mock-up Interface for Model Exchange, Co-Simulation, and Scheduled Execution*. https://fmi-standard.org/.

Gomes, Cláudio, Levi Lucio and Hans Vangheluwe (2019). "Semantics of Co-Simulation Algorithms with Simulator Contracts". In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. Munich, Germany: IEEE, pp. 784–789. DOI: 10.1109/MODELS-C.2019.00124.

Gomes, Cláudio, Bart Meyers et al. (2018). "Semantic Adaptation for FMI Co-Simulation with Hierarchical Simulators". In: *SIMULATION* 95.3, pp. 1–29. DOI: 10.1177/0037549718759775.

Gomes, Cláudio, Bentley James Oakes et al. (2019). "HintCO - Hint-Based Configuration of Co-Simulations". In: *International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. Prague, Czech Republic, pp. 57–68. ISBN: 978-989-758-381-0. DOI: 10.5220/0007830000570068.

Gomes, Cláudio, Casper Thule, David Broman et al. (2018). "Co-Simulation: A Survey". In: *ACM Computing Surveys* 51.3, 49:1–49:33. DOI: 10.1145/3179993.

Gomes, Cláudio, Casper Thule, Levi Lúcio et al. (2020). "Generation of Co-Simulation Algorithms Subject to Simulator Contracts". en. In: *Software Engineering and Formal Methods*. Ed. by Javier Camara and Martin Steffen. Vol. 12226. Lecture Notes in Computer Science. Oslo, Norway: Springer International Publishing, pp. 34–49. ISBN: 978-3-030-57505-2 978-3-030-57506-9. DOI: 10.1007/978-3-030-57506-9_4.

Gravouil, A. and A Combescure (2001). "Multi-time-step explicit–implicit method for non-linear structural dynamics". In: *Int. J. Numer. Methods* 50 (1), pp. 199–225.

Hovmand, M., G. Abbiati and L. V. Andersen (2021). "Real-Time Hybrid Simulation with Nonlinear Numerical Substructures Based on State-Space Modeling". In: *17th World Conference on Earthquake Engineering*. Sendai, Japan, submitted.

Hovmand, Martin, Giuseppe Abbiati and Lars Vabbersgaard Andersen (2021). "Real-time hybrid simulation with nonlinear numerical substructures based on state-space modeling". In: *17 World Conference of Earthquake Engineering (17WCEE)*. Sendai, Japan.

Idinyang, Solomon et al. (2019-07). "Real-time data coupling for hybrid testing in a geotechnical centrifuge". en. In: *International Journal of Physical Modelling in Geotechnics* 19.4, pp. 208–220. ISSN: 1346-213X, 2042-6550. DOI: 10.1680/jphmg.17.00063. URL: https://www.icevirtuallibrary.com/doi/10.1680/jphmg.17.00063 (visited on 2020-03-07).

Kübler, R. and W. Schiehlen (2000). "Two Methods of Simulator Coupling". In: *Mathematical and Computer Modelling of Dynamical Systems* 6.2, pp. 93–113. ISSN: 1387-3954. DOI: 10.1076/1387-3954(200006)6:2;1-M;FT093.

Legaard, Christian Møldrup et al. (2020). "Rapid Prototyping of Self-Adaptive-Systems Using Python Functional Mockup Units". In: *2020 Summer Simulation Conference*. SummerSim '20. Virtual event: ACM New York, NY, USA, to appear.

Lelarasmee, E., Albert E. Ruehli and A. L. Sangiovanni-Vincentelli (1982). "The Waveform Relaxation Method for Time-Domain Analysis of Large Scale Integrated Circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. Vol. 1, pp. 131–145. ISBN: 0278-00701. DOI: 10.1109/TCAD.1982.1270004.

McCrum, D.P. and M.S. Williams (2016-07). "An overview of seismic hybrid testing of engineering structures". en. In: *Engineering Structures* 118, pp. 240–261. ISSN: 01410296. DOI: 10.1016/j.engstruct.2016.03.039. URL: https://linkinghub.elsevier.com/retrieve/pii/S0141029616300748 (visited on 2020-03-05).

McKenna, Frank, Michael H Scott and Gregory L Fenves (2010). "Nonlinear Finite-Element Analysis Software Architecture Using Object Composition". en. In: *Journal of Computing in Civil Engineering* 24.1, p. 13.

Molina, Francisco J. et al. (2011-07). "Monitoring Damping in Pseudo-Dynamic Tests". en. In: *Journal of Earthquake Engineering* 15.6, pp. 877–900. ISSN: 1363-2469, 1559-808X. DOI: 10.1080/13632469.2010.544373. URL: http://www.tandfonline.com/doi/full/10.1080/13632469.2010.544373 (visited on 2020-03-05).

Nakashima, Masayoshi (2020-04). "Hybrid simulation: An early history". en. In: *Earthquake Engineering & Structural Dynamics* 49.10, pp. 949–962. ISSN: 00988847. DOI: 10.1002/eqe.3274. URL: http://doi.wiley.com/10.1002/eqe.3274 (visited on 2020-04-28).

Newton, Arthur Richard and Alberto L. Sangiovanni-Vincentelli (1983-09). "Relaxation-Based Electrical Simulation". In: *SIAM Journal on Scientific and Statistical Computing* 4.3, pp. 485–524. ISSN: 0196-5204. DOI: 10.1137/0904036.

Oakes, Bentley James et al. (2021). "Hint-Based Configuration of Co-Simulations with Algebraic Loops". en. In: *Simulation and Modeling Methodologies, Technologies and Applications*. Vol. 1260. Cham: Springer International Publishing, pp. 1–28. ISBN: 978-3-030-55866-6 978-3-030-55867-3. DOI: 10.1007/978-3-030-55867-3_1.

Pan, Peng, Tao Wang and Masayoshi Nakashima (2016). *Development of online hybrid testing: theory and applications to structural engineering*. en. Oxford [England] ; Waltham, MA: Elsevier / Butterworth Heinemann. ISBN: 978-0-12-803378-4.

Pegon, Pierre and Georges Magonette (2002). *Continuous PsD testing with nonlinear substructuring: presentation of a parallel inter-field procedure*. Tech. rep. I.02.167. Ispra, Italy: European Laboratory for Structural Assessment, Institute for the Protection and the Security of the Citizen, Joint Research Centre.

Sauca, A. et al. (2021-05). “Experimental validation of a hybrid fire testing framework based on dynamic relaxation”. en. In: *Fire Safety Journal* 121, p. 103315. ISSN: 03797112. DOI: 10.1016/j.firesaf.2021.103315. URL: https://linkinghub.elsevier.com/retrieve/pii/S0379711221000552 (visited on 2021-04-09).

Sauder, Thomas et al. (2016-06). “Real-Time Hybrid Model Testing of a Braceless Semi-Submersible Wind Turbine: Part I — The Hybrid Approach”. en. In: *Volume 6: Ocean Space Utilization; Ocean Renewable Energy*. Busan, South Korea: American Society of Mechanical Engineers, V006T09A039. ISBN: 978-0-7918-4997-2. DOI: 10.1115/OMAE2016-54435. URL: https://asmedigitalcollection.asme.org/OMAE/proceedings/OMAE2016/49972/Busan,%20South%20Korea/281288 (visited on 2020-05-01).

Schellenberg, Andreas, Stephen A. Mahin and Gregory L. Fenves (2007-10). “A Software Framework for Hybrid Simulation of Large Structural Systems”. en. In: *Structural Engineering Research Frontiers*. Long Beach, California, United States: American Society of Civil Engineers, pp. 1–16. ISBN: 978-0-7844-0944-2. DOI: 10.1061/40944(249)3. (Visited on 2020-03-04).

Stojadinovic, Bozidar, Gilberto Mosqueda and Stephen A. Mahin (2006-01). “Event-Driven Control System for Geographically Distributed Hybrid Simulation”. en. In: *Journal of Structural Engineering* 132.1, pp. 68–77. ISSN: 0733-9445, 1943-541X. DOI: 10.1061/(ASCE)0733-9445(2006)132:1(68). (Visited on 2020-03-04).

NeuralFMU: Towards Structural Integration of FMUs into Neural Networks

Tobias Thummerer   Josef Kircher   Lars Mikelsons

Chair of Mechatronics, Augsburg University, Germany, {tobias.thummerer@informatik, josef.kircher@student, lars.mikelsons@informatik}.uni-augsburg.de

Abstract

This paper covers two major subjects: First, the presentation of a new open-source library called *FMI.jl* for integrating FMI into the Julia programming environment by providing the possibility to load, parameterize and simulate FMUs. Further, an extension to this library called *FMIFlux.jl* is introduced, that allows the integration of FMUs into a neural network topology to obtain a *NeuralFMU*. This structural combination of an industry typical black-box model and a data-driven machine learning model combines the different advantages of both modeling approaches in one single development environment. This allows for the usage of advanced data driven modeling techniques for physical effects that are difficult to model based on first principles.

Keywords: NeuralFMU, FMI, FMU, Julia, NeuralODE

1 Introduction

Models inside closed simulation tools make hybrid modeling difficult, because for training data-driven model parts, determination of the loss gradient through the Neural Network (NN) and the model itself is needed. Nevertheless, the structural integration of models inside machine learning topologies like NNs is a research topic that gathered more and more attention. When it comes to learning system dynamics, the structural injection of algorithmic numerical solvers into NNs lead to large improvements in performance, memory cost and numerical precision over the use of residual neural networks (Chen et al. 2018), while offering a new range of possibilities, e.g. fitting data that was observed at irregular time steps (Innes et al. 2019). The result of integrating a numerical solver for ordinary differential equations (ODEs) into a NN is known as *NeuralODE*. For the Julia programming language (from here on simply referred to as *Julia*), a ready-to-use library for building and training NeuralODEs named *DiffEqFlux.jl*[1] is already available (Rackauckas, Innes, et al. 2019). Probably the most mentioned point of criticism regarding NeuralODEs is the difficult porting to real world applications (s. section 3.1.2 and 3.1.3).

A different approach for hybrid modeling, as in Raissi, Perdikaris, and Karniadakis (2019), is the integration of the physical model into the machine learning process by evaluating (forward propagating) the physical model as part of the loss function during training in so called Physics-informed Neural Networks (PINNs). In contrast, this contribution focuses on the structural integration of Functional Mock-up Units (FMUs) into the NN itself and *not* only the cost function, allowing much more flexibility with respect to what can be learned and influenced. However, it is also possible to build and train PINNs with the presented library.

Finally, another approach are Bayesian Neural Stochastic Differential Equations (BNSDE) as is Haussmann et al. (2021), which use bayesian model selection together with Probably Approximately Correct (PAC) bayesian bounds during the NN training to improve hybrid model accuracy on basis of noisy prior knowledge. For an overview on the growing field of hybrid modeling see e.g. (Willard et al. 2020) or (Rai and Sahu 2020).

To conclude, hybrid modeling with its different facets is an emerging research field, but still chained to academic use-cases. It seems a logical next step to open up these auspicious ML-technologies, besides many more not mentioned, to industrial applications. Combining physical and data-driven models inside a single industry tool is currently not possible, therefore it is necessary to port models to a more suitable environment. An industry typical model exchange format is needed. Because the Functional Mock-up Interface (FMI) is an open standard and widely used in industry as well as in research applications, it is a suitable candidate for this aim. Finally, a software interface that integrates FMI into the ML-environment is necessary. Therefore, we present two open-source software libraries, which offer all required features:

- *FMI.jl*: load, instantiate, parameterize and simulate FMUs seamlessly inside the Julia programming language
- *FMIFlux.jl*: place FMUs simply inside any feed-forward NN topology and still keep the resulting hybrid model trainable with a standard Automatic Differentiation (AD) training process

Because the result of integrating a numerical solver into a NN is known as *NeuralODE*, we suggest to pursue this naming convention by presenting the integration of a FMU, NN and a numerical solver as *NeuralFMU*.

[1] https://github.com/SciML/DiffEqFlux.jl

DOI 10.3384/ecp21181297

By providing the libraries *FMI.jl* (`https://github.com/ThummeTo/FMI.jl`) and *FMIFlux.jl* (`https://github.com/ThummeTo/FMIFlux.jl`), we want to open the topic *NeuralODEs* for industrial applications, but also lay a foundation to bring other state-of-the-art ML-technologies closer to production. In the following two subsections, short style explanations of the involved tools and techniques are given.

1.1 Julia Programming Language

In this section, it is shortly explained and motivated why the authors picked the Julia programming language for the presented task. Julia is a dynamic typing language developing since 2009 and first published in 2012 (Bezanson, Karpinsky, et al. 2012), with the aim to provide fast numerical computations in a platform-independent, high-level programming language (Bezanson, Edelman, et al. 2015). The language and interpreter was originally invented at the *Massachusetts Institute of Technology*, but since today many other universities and research facilities joined the development of language expansions, which mirrors in many contributions from different countries and even in its own conference, the *JuliaCon*[2]. In Elmqvist, Neumayr, and Otter (2018), the library expansion *Modia.jl*[3] was introduced. *Modia.jl* allows object-orientated white-box modeling of mechanical and electrical systems, syntactically similar to *Modelica*, in Julia.

1.2 Functional Mock-up Interface (FMI)

The FMI-standard (Modelica Association 2020) allows the distribution and exchange of models in a standardized format and independent of the modeling tool. An exported model container, that fulfills the FMI-requirements is called FMU. FMUs can be used in other simulation environments or even inside of entire co-simulations like System Structure and Parameterization (SSP) (Modelica Association 2019). FMUs are subdivided into two major classes: model exchange (ME) and co-simulation (CS). The different use-cases depend on the FMU-type and the availability of standardized, but optional implemented FMI-functions.

This paper is further structured into four topics: The presentation of our libraries *FMI.jl* and *FMIFlux.jl*, an example handling a NeuralFMU setup and training, the explanation of the methodical background and finally a short conclusion with future outlook.

2 Presenting the Libraries

Our Julia-library *FMI.jl* provides high-level commands to unzip, allocate, parameterize and simulate entire FMUs, as well as plotting the solution and parsing model meta data from the model description. Because FMI has already two released specification versions and is under ongoing development[4], one major goal was to provide the ability to simulate different version FMUs with the same user front-end. To satisfy users who prefer close-to-specification programming, as well as users that are new to the topic and favor a smaller but more high-level command set, we provide high-level Julia commands, but also the possibility to use the more low-level commands specified in the FMI-standard (Modelica Association 2020).

2.1 Simulating FMUs

Figure 1. Logo of the library *FMI.jl*.

The shortest way to load a FMU with *FMI.jl*, simulate it for $t \in \{0, 10\}$, gather and plot simulation data for the example variable *mass.s* and free the allocated memory is implemented only by a few lines of code as follows:

Listing 1. Simulating FMUs with *FMI.jl* (high-level).

```julia
using FMI
myFMU = fmiLoad("path/to/myFMU.fmu")
fmiInstantiate!(myFMU)
simData = fmiSimulate(myFMU, 0.0,
   10.0; recordValues=["mass.s"])
fmiPlot(simData)
fmiUnload(myFMU)
```

Please note, that these six lines are not only a code snippet, but a fully runnable Julia program.

For users, that prefer more control over memory and performance, the original C-language command set from the FMI-specification is wrapped into low-level commands and is available, too. A code snippet, that simulates a CS-FMU, looks like this:

Listing 2. Simulating CS-FMUs with *FMI.jl* (low-level).

```julia
using FMI
myFMU = fmiLoad("path/to/myFMU.fmu")
fmuComp = fmiInstantiate!(myFMU)
fmiSetupExperiment(fmuComp, 0.0,
   10.0)
fmiEnterInitializationMode(fmuComp)
fmiExitInitializationMode(fmuComp)
dt = 0.1
ts = 0.0:dt:10.0
for t in ts
  fmiDoStep(fmuComp, t, dt)
end
```

[2] `http://www.juliacon.org`

[3] `https://github.com/ModiaSim/Modia.jl`

[4] The current version is 2.0.2, but an alpha version 3.0 is already available.

```
fmiTerminate(fmuComp)
fmiFreeInstance!(fmuComp)
fmiUnload(myFMU)
```

Note, that these function calls are not dependent on the FMU-Version, but are inspired by the command set of FMI 2.0.2. The underlying FMI-Version is determined in the call `fmiLoad`. Because the naming convention could change in future versions of the standard, version-specific function calls like `fmi2DoStep` (the "2" stands for the FMI-versions 2.x) are available, too. Readers that are familiar with FMI will notice, that the functions `fmiLoad` and `fmiUnload` are not mentioned in the standard definition. The function `fmiLoad` handles the creation of a temporary directory for the unpacked data, unpacking of the FMU-archive, as well as the loading of the FMU-binary and its model description. In `fmiUnload`, all FMU-related memory is freed and the temporary directory is deleted. Beside CS-FMUs, ME-FMUs are supported, too. The numerical solving and event handling for ME-FMUs is performed via the library *DifferentialEquations.jl*[5], the standard library for solving different types of differential equations in Julia (Rackauckas, Singhvi, et al. 2021).

2.2 Integrating FMUs into NNs

Figure 2. Logo of the library extension *FMIFlux.jl*.

The open-source library extension *FMIFlux.jl* allows for the fusion of a FMU and a NN. As in many other machine learning frameworks, a deep NN in Julia using *Flux.jl*[6] is configured by chaining multiple neural layers together. Probably the most intuitive way of integrating a FMU into this topology, is to simply handle the FMU as a network layer. In general, *FMIFlux.jl* does not make restrictions to ...

- ... which FMU-signals can be used as layer inputs and outputs. It is possible to use any variable that can be set via `fmiSetReal` or `fmiSetContinuousStates` as input and any variable that can be retrieved by `fmiGetReal` or `fmiGetDerivatives` as output.
- ... where to place FMUs inside the NN topology, as long as all signals are traceable via AD (no signal cuts).

Dependent on the FMU-type, ME or CS, different setups for NeuralFMUs should be considered. In the following, two possibilities are presented.

2.2.1 ME-FMUs

For most common applications, the use of ME-FMUs will be the first choice. Because of the absence of an integrated numerical solver inside the FMU, there are much more possibilities when it comes to learning dynamic processes. A mathematical view on a ME-FMU leads to the state space equation (Equation 1) and output equation (Equation 2), meaning a ME-FMU computes the state derivative $\dot{\vec{x}}_{me}$ and output values $\vec{y}_{me}$ for the current time step t from a given state $\vec{x}_{me}$ and optional input $\vec{u}_{me}$:

$$\dot{\vec{x}}_{me} = \vec{f}_{me}(\vec{x}_{me}, \vec{u}_{me}, t) \tag{1}$$

$$\vec{y}_{me} = \vec{g}_{me}(\vec{x}_{me}, \vec{u}_{me}, t) \tag{2}$$

Different interfaces between the FMU layer and NN are possible. For example, the number of FMU layer inputs could equal the number of FMU layer outputs and be simply the number of model states. In Figure 3, the visualization of the suggested structure is given. The top NN is fed by the current system state $\vec{x}_{nn}$. The NN is able to learn and compensate state-dependent modeling failures like measurement offsets or physical displacements and thresholds. After that, the corrected state vector $\vec{x}_{me}$ is passed to the ME-FMU, and the current state derivative $\dot{\vec{x}}_{me}$ is retrieved. The bottom NN is able to learn additional physical effects, like friction or other forces, from the state derivative vector. Finally the corrected state derivatives $\dot{\vec{x}}_{nn}$ are integrated by the numerical solver, to retrieve the next system state $\vec{x}_{nn}(t+h)$. Note, that the time step size h can be determined by a modern numerical solver like *Tsit5* (Tsitouras 2011), with different advantages like dynamic step size and order adaption. This is a significant advantage in performance and memory cost over the use of recurrent NNs for numerical integration.

Note, that many other configurations for setting up the NeuralFMU are thinkable, e.g.:

- the top NN could additionally generate a FMU input $\vec{u}_{me}$
- the bottom NN could learn from states $\vec{x}_{me}$ or derivatives $\dot{\vec{x}}_{me}$, for a targeted expansion of the model by additional model equations
- the bottom NN could learn from the FMU output $\vec{y}_{me}$ or other model variables, that can be retrieved by `fmiGetReal`
- there could be a bypass around the FMU between both NNs to forward state-dependent signals from the top NN to the bottom NN
- of course, there is no restriction to fully-connected (dense) layers, other feed-forward layers or even drop-outs are possible

[5] https://diffeq.sciml.ai/stable/
[6] https://fluxml.ai/Flux.jl/stable/

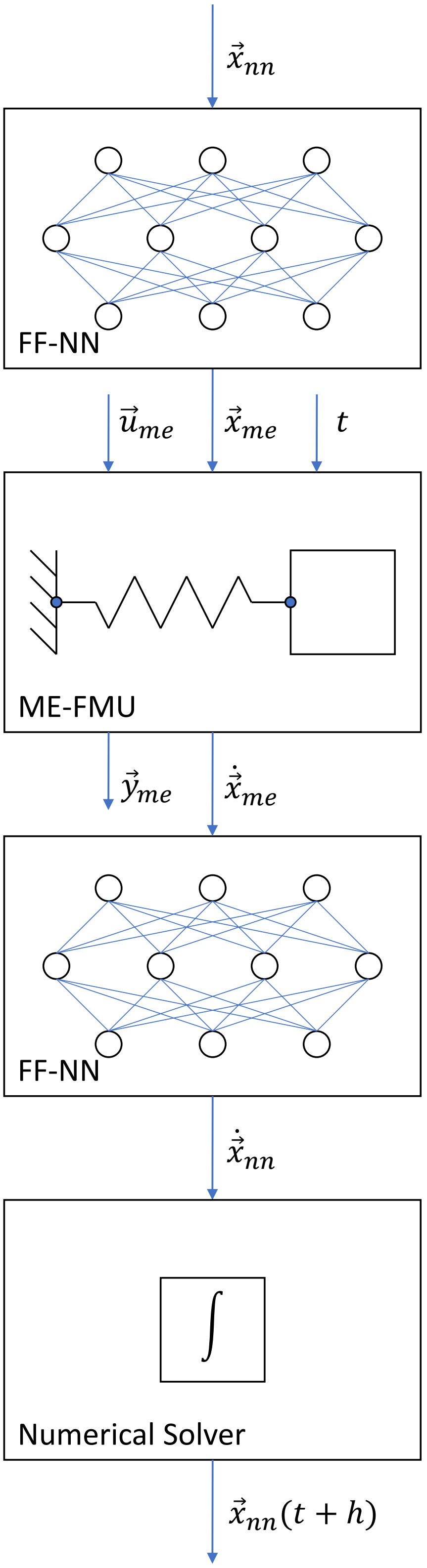


Figure 3. Example for a NeuralFMU (ME).

To implement the presented ME-NeuralFMU in Julia, the following code is sufficient:

Listing 3. A NeuralFMU (ME) in Julia.

```
net = Chain(
 Dense(length(x_nn), ...),
 ...,
 Dense(..., length(x_me)),
 x_me -> fmiDoStepME(myFMU, x_me),
 Dense(length(dx_me), ...),
 ...,
 Dense(..., length(dx_nn)))
nfmu = ME_NeuralFMU(net, ...)
```

2.2.2 CS-FMUs

Beside ME-FMUs, it is also possible to use CS-FMUs as part of NeuralFMUs. For CS-FMUs, a numerical solver like *CVODE* (Hindmarsh, Serban, et al. 2021) is already integrated and compiled as part of the FMU itself.

This prevents the manipulation of system dynamics at the most effective point: Between the FMU state derivative output and the numerical integration. However, other tasks like learning an error correction term, are still possible to implement. The presence of a numerical solver leads to a different mathematical description compared to ME-FMUs: The CS-FMU computes the next state $\vec{x}_{cs}(t+h)$ and output $\vec{y}_{cs}(t+h)$ dependent on its internal current state $\vec{x}_{cs}$ (Eq. 3 and 4). Unlike for ME, the state and derivative values of a CS-FMU are not necessarily known (disclosed via FMI).

$$\vec{x}_{cs}(t+h) = \vec{f}_{cs}(\vec{x}_{cs}, \vec{u}_{cs}, t, h) \tag{3}$$
$$\vec{y}_{cs}(t+h) = \vec{g}_{cs}(\vec{x}_{cs}, \vec{u}_{cs}, t, h) \tag{4}$$

In case of CS-FMUs, the number of layer inputs could be based on the number of FMU-inputs and the number of outputs in analogy. As for ME-NeuralFMUs, this is just one possible setup for a CS-NeuralFMU. Figure 4 shows the topology of the considered NeuralFMU. The top NN retrieves an input signal $\vec{u}_{nn}$, which is corrected into $\vec{u}_{cs}$. Note, that here training of the top NN is only possible if the FMU output is sensitive to the FMU input. This is often not the case for physical simulations. Inside the CS-FMU, the input $\vec{u}_{cs}$ is set, an integrator step with step size h is performed and the FMU output $\vec{y}_{cs}(t+h)$ is forwarded to the bottom NN. Here, a simple error correction into $\vec{y}_{nn}(t+h)$ is performed, meaning the error is determined and compensated without necessarily learning the mathematical representation of the underlying physical effect.

Note that for CS, even if the macro step size h must be determined by the user, it does not need to be constant if the numerical solver inside the FMU supports varying step sizes. If so, the internal solver step size may vary from h, in fact h acts as a upper boundary for the internal micro step size. As a result, if the FMU is compiled with a variable-step solver, unnecessarily small values for

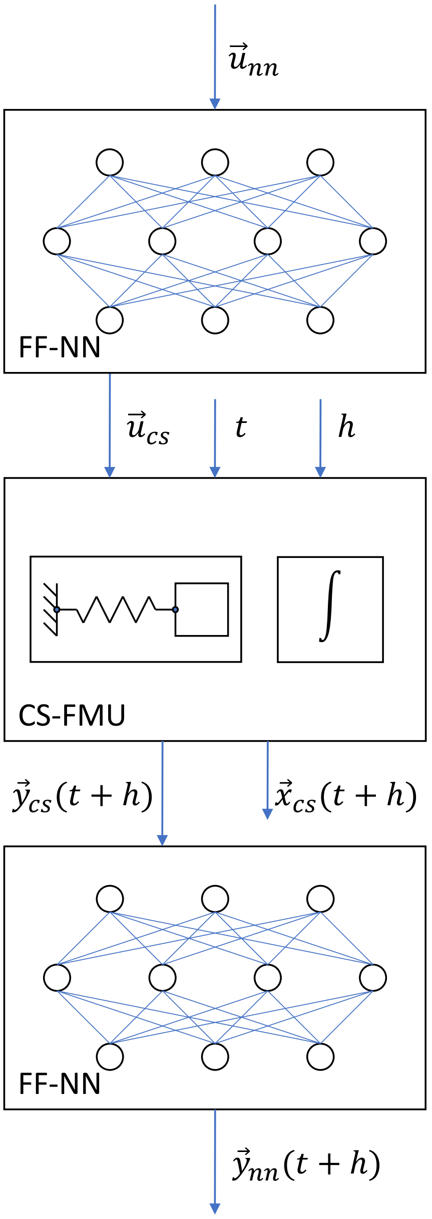


Figure 4. Example for integrating a CS-FMU into a NN.

h will have negative influence on the internal solver performance, but large values will not destabilize the numerical integration.

Other configurations for setting up the hybrid structure are interesting, e.g.:

- the bottom NN could learn from the system state $\vec{x}_{cs}(t+h)$ or state derivative $\dot{\vec{x}}_{cs}(t+h)$
- the step size h could be learned by an additional NN to optimize simulation performance
- there could be a bypass around the FMU between both NNs to forward input-dependent signals from the top NN to the bottom NN

The software implementation of the considered CS-NeuralFMU looks as follows:

Listing 4. A NeuralFMU (CS) in Julia.

```
net = Chain(
 Dense(length(u_nn), ...),
 ...,
 Dense(..., length(u_cs)),
 u_cs -> fmiInputDoStepCSOutput(
    myFMU, h, u_cs),
 Dense(length(y_cs), ...),
 ...,
 Dense(..., length(y_nn)))
nfmu = CS_NeuralFMU(net, ...)
```

First, the function `fmiInputDoStepCSOutput` sets all FMU-inputs to the values `u`, respectively the output of the previous net layer. After that, a `fmiDoStep` with step size `h` is performed and finally the FMU output is retrieved and passed to the next layer. Because of the integration over time inside the CS-FMU to retrieve new system states, it is necessary to reset the FMU for every training run, similar to the training of recurrent NNs.

3 Methodical Background

High-performance machine learning frameworks like *Flux.jl* are using AD for reverse-mode differentiation through the NN topology. Because all mathematical operations inside the NN are known (white-box), this is a very efficient way to derive the gradient and train NNs. On the other hand, the jacobian over a black-box FMU is unknown from the view of an AD framework, because the model structure is (in general) hidden as compiled machine code. This jacobian is part of the loss gradient AD-chain and needs to be determined.

Beside others, the most common and default AD framework in Julia is *Zygote.jl*[7]. A remarkable feature of *Zygote.jl* is, that it provides the ability to define custom adjoints, meaning the gradient of a custom function can be defined to be an arbitrary function. This renders the possibility to pass a custom gradient for a FMU-representation to the AD-framework, which will be later used to derive the total loss function gradient during the training process.

3.1 Gradient of the Loss Function

For the efficient training of NNs, the gradient of the loss function according to the net parameters (weights and biases) is needed. In the following, three methods to derive the loss gradient will be discussed: AD, forward sensitivity analysis and backward adjoints. In the following, only ME-NeuralFMUs are considered and the loss function $l(\vec{x}_{nn}(\vec{p}))$ is expected to depend explicitly on the system state $\vec{x}_{nn}$ (the NeuralFMU output) and only implicitly on the net parameters $\vec{p}$.

[7] `https://fluxml.ai/Zygote.jl/latest/`

3.1.1 Automatic Differentiation (AD)

For white-box systems, like native implemented numerical solvers, one possible approach to provide the gradient is AD (Rackauckas, Innes, et al. 2019). In general, the mathematical operations inside a FMU are not known (compiled binary), meaning despite AD being a very common technique, it is only suitable for determining the gradient of the NN, but not the jacobian over a FMU. The unknown jacobian $\vec{J}_{fmu}$ over the FMU layer with layer inputs $\vec{u}$ and outputs $\vec{y}$ is noted as follows:

$$\vec{J}_{fmu} = \frac{\partial \vec{y}}{\partial \vec{u}} \tag{5}$$

Inserting $\vec{u} = \vec{x}_{me}$ and $\vec{y} = \dot{\vec{x}}_{me}$ results in the jacobian $\vec{J}_{me}$ for the FMU from the example in subsection 2.2.1 (ME), inserting $\vec{u} = \vec{u}_{cs}$ and $\vec{y} = \vec{y}_{cs}(t+h)$ results in the jacobian matrix $\vec{J}_{cs}$ for subsection 2.2.2 (CS):

$$\vec{J}_{me} = \frac{\partial \dot{\vec{x}}}{\partial \vec{x}} \tag{6}$$

$$\vec{J}_{cs} = \frac{\partial \vec{y}(t+h)}{\partial \vec{u}(t)} \approx \frac{\partial \vec{y}(t)}{\partial \vec{u}(t)} \tag{7}$$

The simplification in Equation 7 does not lead to problems for small step sizes h, because in practice, a small error in the jacobian only negatively affects the optimization performance (convergence speed) and not the convergence itself. However, the quantity of the mentioned error is dependent on the the optimization algorithm and parameterization and h should be selected on the basis of expert knowledge about the model and optimizer or - if not available - as part of hyper parameter tuning.

3.1.2 Forward Sensitivities

To retrieve the partial derivative (sensitivity) of the system state according to a net parameter $p_i \in \vec{p}$ and thus in straight forward manner also the gradient of the loss function, another common approach is Forward Sensitivity Analysis. Sensitivities can be estimated by extending the system state by additional sensitivity equations in form of ODEs. Dependent on the number of parameters $|\vec{p}|$, this leads to large ODE systems of size $(1+|\vec{p}|) \cdot |\vec{x}|$ (Hindmarsh and Serban 2006, p. 21) and therefore worsens the overall computation and memory performance. Computations can be reduced, but at a higher memory cost (Rackauckas, Innes, et al. 2019, p. 15). For a ME-NeuralFMU, the sensitivity equation for a parameter p_i can be formulated as in Hindmarsh and Serban (2006, p. 19):

$$\frac{d}{dt}\frac{\partial \vec{x}_{nn}}{\partial p_i} = \underbrace{\frac{\partial \dot{\vec{x}}_{nn}}{\partial \vec{x}_{nn}}}_{\vec{J}_{nn}} \cdot \frac{\partial \vec{x}_{nn}}{\partial p_i} + \frac{\partial \dot{\vec{x}}_{nn}}{\partial p_i} \tag{8}$$

The jacobian of the entire NeuralFMU $\vec{J}_{nn}$ can be described via chain-rule as a product of the three jacobians $\vec{J}_{bottom}$ (over the bottom part of the NN), $\vec{J}_{me}$ (over the ME-FMU) and $\vec{J}_{top}$ (over the top part of the NN):

$$\underbrace{\frac{\partial \dot{\vec{x}}_{nn}}{\partial \vec{x}_{nn}}}_{\vec{J}_{nn}} = \underbrace{\frac{\partial \dot{\vec{x}}_{nn}}{\partial \dot{\vec{x}}_{me}}}_{\vec{J}_{bottom}} \cdot \underbrace{\frac{\partial \dot{\vec{x}}_{me}}{\partial \vec{x}_{me}}}_{\vec{J}_{me}} \cdot \underbrace{\frac{\partial \vec{x}_{me}}{\partial \vec{x}_{nn}}}_{\vec{J}_{top}} \tag{9}$$

Inserting Equation 9 into Equation 8 yields:

$$\frac{d}{dt}\frac{\partial \vec{x}_{nn}}{\partial p_i} = \underbrace{\frac{\partial \dot{\vec{x}}_{nn}}{\partial \dot{\vec{x}}_{me}}}_{\vec{J}_{bottom}} \cdot \underbrace{\frac{\partial \dot{\vec{x}}_{me}}{\partial \vec{x}_{me}}}_{\vec{J}_{me}} \cdot \underbrace{\frac{\partial \vec{x}_{me}}{\partial \vec{x}_{nn}}}_{\vec{J}_{top}} \cdot \underbrace{\frac{\partial \vec{x}_{nn}}{\partial p_i}}_{\vec{g}_{top\_i}} + \underbrace{\frac{\partial \dot{\vec{x}}_{nn}}{\partial p_i}}_{\vec{g}_{bottom\_i}} \tag{10}$$

Retrieving the jacobian $\vec{J}_{me}$ is handled in subsection 3.2, the jacobians $\vec{J}_{bottom}$ and $\vec{J}_{top}$ are determined by AD, because the NN is modeled as white-box and all mathematical operations are known. The remaining gradients, $\vec{g}_{top\_i}$ and $\vec{g}_{bottom\_i}$ can be determined building an AD-chain, dependent on the parameter locations inside the NN (top or bottom part), the jacobian $\vec{J}_{me}$ is needed.

As mentioned, the poor scalability with parameter count makes forward sensitivities unattractive for ML-applications with large parameter spaces, but it remains an interesting option for small NNs. To decide which sensitivity approach to pick for a specific NN size, a useful comparison according to performance of forward and other sensitivity estimation techniques, dependent on the number of involved parameters, can be found in (Rackauckas, Ma, et al. 2018).

3.1.3 Backward Adjoints

The performance disadvantage of Forward Sensitivity Analysis motivates the search for a method, that scales better with a high parameter count. Retrieving the directional derivatives over a black-box FMU sounds similar to the reverse-mode differentiation over a black-box numerical solver as in Chen et al. (2018). The name *Backward Adjoints* results from solving the ODE adjoint problem backwards in time:

$$\vec{a} = \frac{dl(\vec{x}_{nn})}{d\vec{x}_{nn}} \tag{11}$$

$$\frac{d\vec{a}}{dt} = -\vec{a}^T \cdot \underbrace{\frac{\partial \dot{\vec{x}}_{nn}}{\partial \vec{x}_{nn}}}_{\vec{J}_{nn}} \tag{12}$$

The jacobian $\vec{J}_{nn}$ can be retrieved like in Equation 9. The searched gradient of the loss function is then given as in Hindmarsh and Serban (2006, p. 22):

$$\frac{dl(\vec{x}_{nn})}{d\vec{p}} = \vec{a}^T(t_0) \cdot \underbrace{\frac{\partial \vec{x}_{nn}}{\partial \vec{p}}}_{\vec{g}_{top}}(t_0) + \int_{t_0}^{t_1} \vec{a}^T(t)\underbrace{\frac{\partial \dot{\vec{x}}_{nn}}{\partial \vec{p}}(t)\,dt}_{\vec{g}_{bottom}} \tag{13}$$

Here $\vec{g}_{top}$ and $\vec{g}_{bottom}$ can be determined again using AD and $\vec{J}_{me}$. To conclude, the backward adjoint ODE system with dimension $|\vec{x}|$ has to be solved only once independent of the number of parameters and therefore requires less computations for large parameter spaces compared to forward sensitivities. On the other hand, backward adjoints are only suitable, if the loss function gradient is smooth and bounded (Hindmarsh and Serban 2006, p. 22), which limits the possible use for this technique to continuous systems and therefore to almost only research applications.

3.2 Jacobian of the FMU

Independent of the chosen method, the jacobian over the FMU $\vec{J}_{me}$ is needed to keep the NeuralFMU trainable, but is unknown and must be determined. In the following, we suggest two possibilities to retrieve the gradient over a FMU: Finite Differences and the use of the built-in function `fmi2GetDirectionalDerivative`.

3.2.1 Finite Differences

The jacobian can be derived by selective input modification, sampling of additional trajectory points and estimating the partial derivatives via finite differences. Note, that this approach is an option for ME-FMUs, for CS-FMUs only if the optional functions to store and set previous FMU states, `fmi2GetState` and `fmi2SetState`, are available. Otherwise, sampling would require to setup a new simulation for every FMU layer input and every considered time step, if the system state vector is unknown. This would be an unacceptable effort for most industrial applications with large models.

3.2.2 Built-in Directional Derivatives

The preferred approach in this paper is different and benefits from a major advantage of the FMI-standard: Fully implemented FMUs provide the partial derivatives between any variable pair, thus the partial derivative between the systems states and derivatives (ME) or the FMU inputs and its outputs (CS) is known at any simulation time step and does *not* need to be estimated by additional methods. In FMI 2.0.2, the partial derivatives can be retrieved by calling the optional function `fmi2GetDirectionalDerivative` (Modelica Association 2020, p. 26). Depending on the underlying implementation of this function, which can vary between exporting tools, this can be a fast and reliable way to gather directional derivatives in fully implemented FMUs.

To conclude, the key step is to forward the directional derivatives over the FMU to the AD-framework *Zygote.jl*. As mentioned, *Zygote.jl* provides a feature to define a custom gradient over any function. In this case, the gradients for the functions `fmiDoStepME` and `fmiInputDoStepCSOutput` are wrapped to calls to `fmi2GetDirectionalDerivative`.

Finally, we provide a seamless link to the ML-library *Flux.jl*, meaning NeuralFMUs can be trained the same way as a convenient NN in Julia:

Listing 5. Training NeuralFMUs in Julia.

```
nfmu = NeuralFMU(net, ...)
p_net = Flux.params(nfmu)
Flux.train!(..., p_net, ...)
```

As a final note, the presented methodical procedure, integrating FMUs into the Julia machine learning environment, can be transferred to other AD-frameworks in other programming languages like *Python*.

4 Example

When modeling physical systems, it's often not practical to model solely based on first principle and parameterize every physical aspect. For example, when modeling mechanical, electrical or hydraulic systems, a typical modeling assumption is the negligence of friction or the use of greatly simplified friction models. Even when using friction models, the parameterization of these is a difficult and error prone task. Therefore, we decided to show the benefits of the presented hybrid modeling technique on an easy to understand example from this set of problems.

4.1 Model

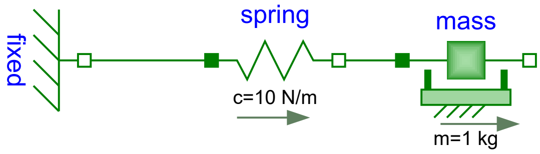


Figure 5. The reference system in *Modelica*.

As in Figure 5, the reference system is modeled as one mass oscillator (horizontal spring-pendulum) with mass m, spring constant c and relaxed spring length s_{rel}, defined by the differential equation:

$$\ddot{s} = \dot{v} = a = \frac{c \cdot (s_0 + s_{rel} - s) - f_{fric}(v)}{m} \tag{14}$$

The parameter s_0 describes the absolute position of the fixed anchor point, allowing to model a system displacement or a constant position measurement offset. Further, the friction force f_{fric} between the pendulum body and the underlying ground is implemented with the non-linear, discontinuous friction model from *MassWithStopAndFriction*[8] as part of the Modelica Standard Library (MSL). The friction term for positive v denotes:

$$f_{fric}(v) = f_{coulomb} + f_{prop} \cdot v + f_{stribeck} \cdot e^{-f_{exp} \cdot v} \tag{15}$$

This friction model consists of parameters for the constant Coulomb-friction $f_{coulomb}$, f_{prop} for the velocity-proportional (viscous) friction and $f_{stribeck}$ for the exponential Stribeck-friction. The FMU (white box) model on the other hand, only covers the modeling of a continuous, frictionless spring pendulum, therefore with $f_{fric}(v) = 0$.

[8] Modelica.Mechanics.Translational.Components. MassWithStopAndFriction (MSL 3.2.3)

The aim here is to learn a generalized representation of the parameterized friction-model in Equation 15 from measurements of the pendulum's movement over time. Further, a displacement of $s_0 = 0.1\,m$ is added to the FMU model (modeling failure), which should be detected and compensated. Both systems are parameterized as in Table 1.

Table 1. Parameterization of the reference and FMU model.

Parameter	*Value* *ref. model*	*Value* *FMU model*	*Unit*
f_{prop}	0.05	0.0	$^{N\cdot s}/_{m}$
$f_{coulomb}$	0.25	0.0	N
$f_{stribeck}$	0.5	0.0	N
f_{exp}	2.0	0.0	$^{s}/_{m}$
$mass.m$	1.0	1.0	kg
$spring.c$	10.0	10.0	$^{N}/_{m}$
$spring.s_{rel}$	1.0	1.0	m
$fixed.s_0$	0.0	0.1	m

4.2 NeuralFMU Setup

We will show, that with a *NeuralFMU*-structure as in Figure 3, it is possible to learn a simplified friction model as well as the constant system displacement (modeling failure) with a simple fully-connected feed-forward NN as in Table 2. The network topology results from a simple random search hyper parameter optimization for a NeuralFMU model with a maximum of 150 net parameters and 8 layers. All weights are initialized with standard-normal distributed random values and all biases with zeros, except the weights of layer #1 are initialized as identity matrix, to start training with a neutral setup and keep the system closer to the preferred intuitive solution. The loss function is defined as simple mean squared error between equidistant sample points of the NeuralFMU and the reference system.

Table 2. Topology of the example NeuralFMU.

Layer	*Inputs*	*Outputs*	*Activation*
#1 (input)	2	2	identity
#2 (FMU)	2	2	none
#3 (hidden)	2	8	identity
#4 (hidden)	8	8	tanh
#5 (output)	8	2	identity

The corresponding code is available online as part of the library repository[9].

4.3 Results

4.3.1 Training

After a short training[10] of 2500 runs on 400 data points (each position and velocity), the hybrid model is able to imitate the reference system on training data, as can be seen in Fig. 6 for position and 7 for velocity. The training has not converged yet, further training will lead to a improved fit. For the training case, the system was initialized with $mass.s_0 = 0.5\,m$ (the pendulum equilibrium is at $1.0\,m$) and $mass.v_0 = 0\,^{m}/_{s}$. Please keep in mind that the NeuralFMU was only trained by data gathered from one single simulation scenario.

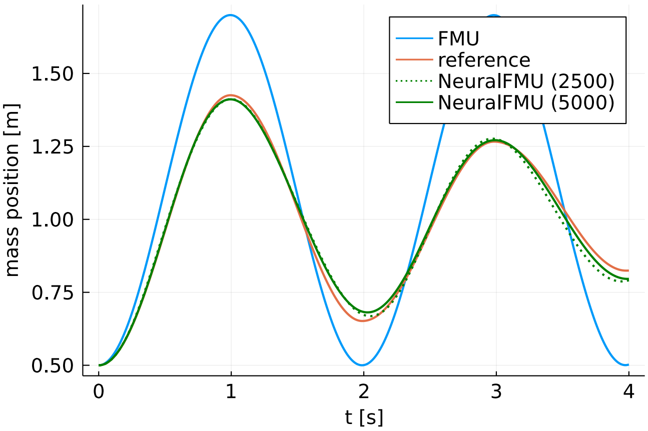


Figure 6. Comparison of the mass position of the FMU, reference system and the NeuralFMU after 2500 and 5000 training steps on training data.

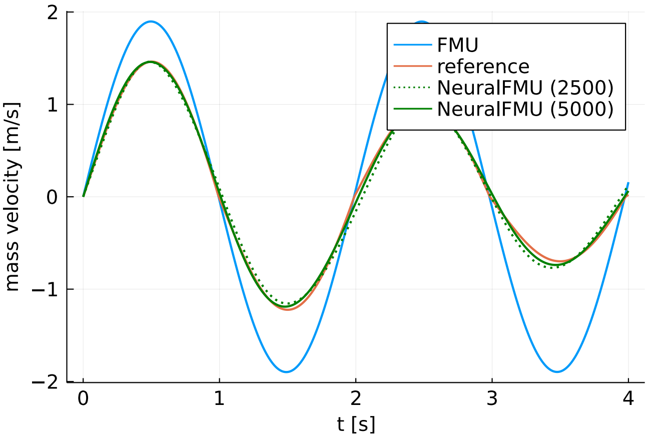


Figure 7. Comparison of the mass velocity of the FMU, reference system and the NeuralFMU after 2500 and 5000 training steps on training data.

[9] https://github.com/ThummeTo/FMIFlux.jl/blob/main/example/modelica_conference_2021.jl

[10] Training was performed single-core on a desktop CPU (Intel® Core™ i7-8565U) and took about 22.5 minutes. GPU training is under development.

4.3.2 Testing

Even if the deviation between NeuralFMU and reference system is larger for testing then for training data, the hybrid model performs well on the test case with a different (untrained) initial system state (Figure 8 and 9). For testing, the system is initialized with $mass.s_0 = 1.0\,m$ and $mass.v_0 = -1.5\,{}^{m}\!/\!{}_{s}$.

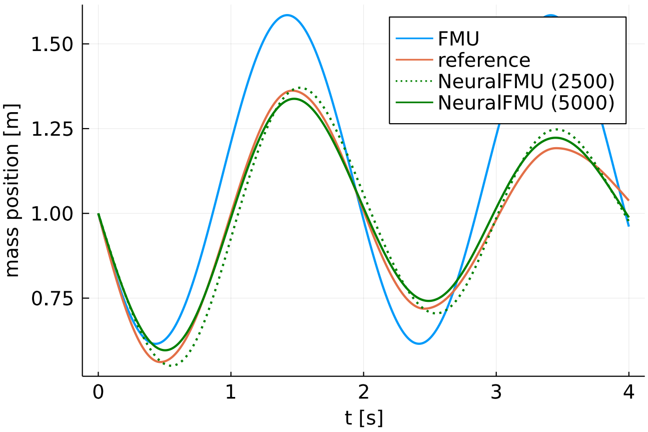


Figure 8. Comparison of the mass position of the FMU, reference system and the NeuralFMU after 2500 and 5000 training steps on testing data.

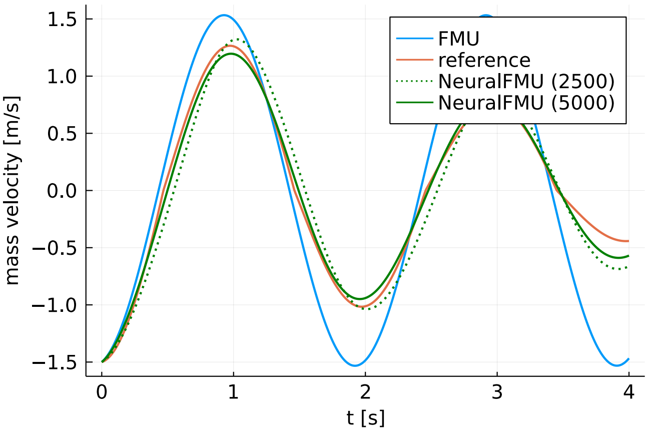


Figure 9. Comparison of the mass velocity of the FMU, reference system and the NeuralFMU after 2500 and 5000 training steps on testing data.

The bottom part of the NN learned the physical effect *discontinuous friction* in a generalized way, because the net was trained based on the state derivatives instead of the states themselves. A comparison of the friction model of the reference system, the FMU and the learned friction model, extracted from the bottom part NN of the NeuralFMU, is shown in Figure 10. The learned friction model is a simplification of the reference friction model, because of the small net layout and a lack of data at the discontinuity near $v = 0$. Finally, also the displacement modeling failure of the white-box model (FMU) was canceled out by the small top NN as can be seen in Figure 11.

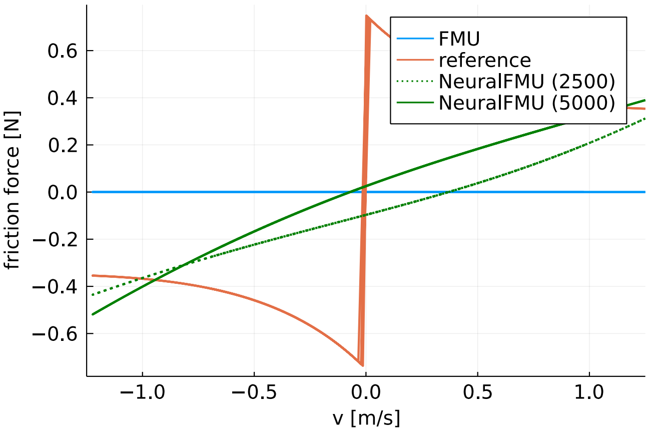


Figure 10. Comparison of the friction models of the FMU, reference system and the NeuralFMU (bottom part NN) after 2500 and 5000 training steps on testing data.

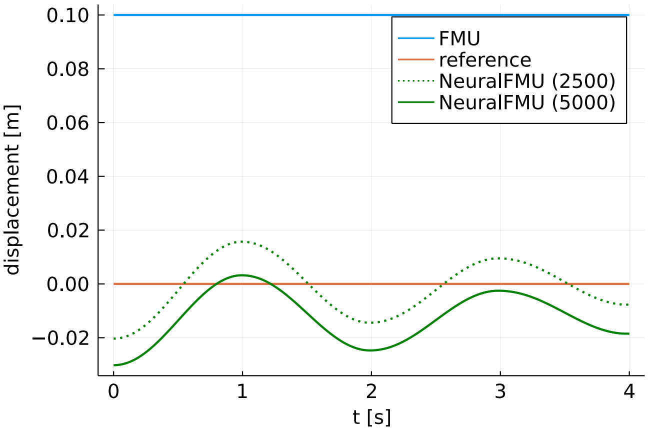


Figure 11. Comparison of the displacements of the FMU, reference system and the NeuralFMU (top part NN) after 2500 and 5000 training steps on testing data.

5 Conclusion

The presented open source library *FMI.jl* (`https://github.com/ThummeTo/FMI.jl`) allows the easy and seamless integration of FMI-models into the Julia programming language. FMUs can be loaded, parameterized and simulated using the abilities of the FMI-standard. Optional functions like retrieving the partial derivatives or manipulating the FMU state are available if supported by the FMU. The library release version 0.1.4 is compatible with FMI 2.0.x (the common version at the time of release), supporting upcoming standard updates like FMI 3.0 is planned. The library currently supports ME- as well as CS-FMUs, running on Windows and Linux operation systems. Event-handling to simulate discontinuous ME-FMUs is supported.

The library extension *FMIFlux.jl* (`https://github.com/ThummeTo/FMIFlux.jl`) makes FMUs differentiable and opens the possibility to setup and train NeuralFMUs, the structural combination of a FMU, NN and a numerical solver. Proper event-handling during backpropagation whilst training of NeuralFMUs is under development, even if there were no problems during training with the discontinuous model from the paper example. A cumulative publication is planned, focusing on a real industrial use-case instead of a methodical presentation.

Current and future work covers the implementation of a more general custom adjoint, meaning despite *Zygote.jl*, other AD-frameworks will be supported. Further, we are working on different fall-backs if the optional function `fmi2GetDirectionalDerivatives` is not available. The finite differences approach for ME-FMUs is already implemented, sampling via `fmi2GetState` and `fmi2SetState` for CS-FMUs will be supported soon.

FMUs contain the model as compiled binary, therefore FMU related computations must be performed on the CPU. On the other hand, deploying NNs on optimized hardware like GPUs often results in a better training performance. Currently, the training of the NeuralFMU is completely done on the CPU. A hybrid hardware training loop with the FMU on the CPU and NN on the GPU may lead to performance improvements for wider and deeper NN-topologies.

An extension of the library to the CS-standard SSP (Modelica Association 2019), including the necessary machine learning back-end, is near completion. This will allow the integration of complete CSs into a NN topology and retrieve a *NeuralSSP*.

Beside NeuralFMUs, *FMIFlux.jl* paves the way for other hybrid modeling techniques and new industrial use-cases by making FMUs differentiable in an AD-framework. The authors are excited about any assistance they can get to extend the library repositories by new features and maintain them for the upcoming technology progress. Contributors are welcome.

Acknowledgments

The authors like to thank *Andreas Heuermann* for his hints and nice feedback during the development of the library. Further, we thank *Florian Schläffer* for designing the beautiful library logos. This work has been partly supported by the ITEA 3 cluster programme for the project UPSIM - Unleash Potentials in Simulation.

References

Bezanson, Jeff, Alan Edelman, et al. (2015). "Julia: A Fresh Approach to Numerical Computing". In: *CoRR* abs/1411.1607. arXiv: 1411.1607. URL: http://arxiv.org/abs/1411.1607.

Bezanson, Jeff, Stefan Karpinsky, et al. (2012). "Julia: A Fast Dynamic Language for Technical Computing". In: *CoRR* abs/1209.5145. arXiv: 1209.5145. URL: http://arxiv.org/abs/1209.5145.

Chen, Tian Qi et al. (2018). "Neural Ordinary Differential Equations". In: *CoRR* abs/1806.07366. arXiv: 1806.07366. URL: http://arxiv.org/abs/1806.07366.

Elmqvist, Hilding, Andrea Neumayr, and Martin Otter (2018). "Modia - Dynamic Modeling and Simulation with Julia". In: *Juliacon 2018*. URL: https://elib.dlr.de/124133/.

Haussmann, Manuel et al. (2021). "Learning Partially Known Stochastic Dynamics with Empirical PAC Bayes". In: arXiv: 2006.09914 [cs.LG].

Hindmarsh, Alan C. and Radu Serban (2006-11). *User Documentation for CVODES v2.5.0*. Tech. rep. URL: https://www.researchgate.net/profile/Radu-Serban/publication/239581887_User_Documentation_for_CVODES_v210/links/00b7d534f0be2a496f000000/User-Documentation-for-CVODES-v210.pdf.

Hindmarsh, Alan C., Radu Serban, et al. (2021-02). *User Documentation for CVODE v5.7.0 (sundials v5.7.0)*. Tech. rep. URL: https://computing.llnl.gov/sites/default/files/cv_guide-5.7.0.pdf.

Innes, Mike et al. (2019). "A Differentiable Programming System to Bridge Machine Learning and Scientific Computing". In: *CoRR* abs/1907.07587. arXiv: 1907.07587. URL: http://arxiv.org/abs/1907.07587.

Modelica Association (2019-03). *System Structure and Parameterization. Document version: 1.0.* Tech. rep. Linköping: Modelica Association. URL: https://ssp-standard.org/publications/SSP10/SystemStructureAndParameterization10.pdf.

Modelica Association (2020-12). *Functional Mock-up InterfaceforModel Exchange and Co-Simulation. Document version: 2.0.2*. Tech. rep. Linköping: Modelica Association. URL: https://github.com/modelica/fmi-standard/releases/download/v2.0.2/FMI-Specification-2.0.2.pdf.

Rackauckas, Christopher, Mike Innes, et al. (2019). "DiffEqFlux.jl - A Julia Library for Neural Differential Equations". In: *CoRR* abs/1902.02376. arXiv: 1902.02376. URL: http://arxiv.org/abs/1902.02376.

Rackauckas, Christopher, Yingbo Ma, et al. (2018). "A Comparison of Automatic Differentiation and Continuous Sensitivity Analysis for Derivatives of Differential Equation Solutions". In: arXiv: 1812.01892 [cs.NA].

Rackauckas, Christopher, Anshul Singhvi, et al. (2021-07). *SciML/DifferentialEquations.jl: v6.17.2*. Version v6.17.2. DOI: 10.5281/zenodo.5069045. URL: https://doi.org/10.5281/zenodo.5069045.

Rai, Rahul and Chandan K. Sahu (2020). "Driven by Data or Derived Through Physics? A Review of Hybrid Physics Guided Machine Learning Techniques With Cyber-Physical System (CPS) Focus". In: *IEEE Access* 8, pp. 71050–71073. DOI: 10.1109/ACCESS.2020.2987324.

Raissi, M., P. Perdikaris, and G.E. Karniadakis (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378, pp. 686–707. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.10.045. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Tsitouras, Ch. (2011). "Runge–Kutta pairs of order 5(4) satisfying only the first column simplifying assumption". In: *Computers & Mathematics with Applications* 62.2, pp. 770–775. ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2011.06.002. URL: https://www.sciencedirect.com/science/article/pii/S0898122111004706.

Willard, Jared et al. (2020). "Integrating Physics-Based Modeling with Machine Learning: A Survey". In: arXiv: 2003.04919 [physics.comp-ph].

Sensitivity Analysis of a Car Shock Absorber Through a Functional Mock-up Units-Based Modelling Strategy

VUILLOD Bruno[1,2] HALLO Ludovic[1] PANETTIERI Enrico[2] MONTEMURRO Marco[2]

[1]Commissariat à l'énergie atomique et aux énergies alternatives (CEA), France, ludovic.hallo@cea.fr

[2]Arts et Métiers Institute of Technology, Université de Bordeaux, CNRS, INRA, Bordeaux INP, HESAM Université, I2M UMR 5295, F-33405 Talence, France, {bruno.vuillod, enrico.panettieri, marco.montemurro}@ensam.eu

Abstract

In Model-Based System Engineering (MBSE), some functional sub-systems can have a considerable influence on the overall system behaviour, whilst the effect of other ones can be neglected. Of course, the former requires a refined modelling approach, whilst the latter can be suitably represented by means of low-fidelity models (usually 0D models). Being capable of identifying the required precision level of sub-systems can help reducing the system complexity, with a negligible impact on the overall accuracy and help deepen the calculations in the system parts where it is necessary.

To determine which sub-systems models must be refined, suitable indicators must be introduced to assess their influence on the global system behaviour. To this purpose, in this work, a sensitivity analysis based on Sobol's indices coupled with a simple mechanical model developed in the Modelica environment is proposed to achieve the aforementioned task.

Keywords: Modelica, Sensitivity Analysis, Sobol's Index, Dynamical System.

1 Introduction

The first step to generate a Modelica model consists of setting the blocks scheme accounting for the global system architecture. The second step, is to enrich this scheme with additional components which include physical-based responses. This task may be difficult to be realised because it needs specific attention and a deep knowledge of multi-physics dynamical problems. Once the functional system architecture is finalised, Modelica can be used to solve it. This modelling approach, when complex systems are analysed, can lead to prohibitive computational costs, poor accuracy and incompatibility with project requirements, see, for instance, the different stages of a V-and-V organisation for example (Plogert 1996).

A viable strategy to improve the modelling approach, i.e. by proposing an efficient model constituting a good balance between accuracy and computational costs, consists in identifying those system parameters influencing the most the considered system response. This task is anything but trivial when using classical software. Modelica can be conveniently employed to achieve this task because it allows for splitting the whole system into sub-systems and allows identifying the main parameters influencing its overall behaviour.

In this paper, a model of a car shock absorber, based on the Functional Mock-up Units (FMU) file export option of Modelica, is developed and integrated in an in-house code which performs the sensitivity analysis. The car shock absorber is based on a classic mass-spring-damper model enhanced with some fundamental notions of hydromechanics. In particular, the proposed model is characterised by 11 input variables and a single output: the equilibrium position of the car shock absorber at the generic time. The interest of using FMU relies on fast multi-physics simulations, with the possibility to easily change the parameters set. Therefore, a statistical analysis (which requires a huge amount of simulation runs) can be carried out by modifying the parameters set through the use of FMU. The sensitivity analysis presented in this paper is based on the use of Sobol's indices (Sobol 1993), which allows for determining the model parameters influencing the most the considered output. Sobol's indices allows expressing the variance of a given output by accounting for the variance of each input variable and for the coupling effects among them (depending on the order of the Sobol's inde). This method is based on Hoeffding decomposition (Hoeffding 1948) and has been adapted to the discrete numerical systems by Saltelli (Saltelli 2002). The sensitivity analysis proposed in this work is carried out through an in-house code called Nebraska, coupled with the Modelica model via its FMU file and a Python code.

Only a few literature references exist on the sensitivity analysis based on Sobol's indices and on the Hoeffding decomposition: (Sobol 1993), (Hoeffding 1948), (Saltelli 2002). Nowadays, these approaches are being reconsidered with the development of parallel and/or distributed computing capabilities of modern computers, which enable a large number of runs in a reasonable time. Some recent works make use of Sobol's indices in different applications: global optimisation strategy based on meta-heuristic algorithms (Janon 2019), metamodelling strategies (Marrel et al. 2009) or extension of the Sobol's index

DOI
10.3384/ecp21181307

theory to the case of dependent input variables (Chastaing, Gamboa, and Prieur 2015).

The paper is organised as follows. In Section 2, the mathematical formulation at the basis of the proposed model is presented together with the fundamentals of the Sobol's analysis. In section 3, the details of the Modelica model are provided, whilst Section 4 presents the numerical results and the related discussion. Section 5 ends the paper with some concluding remarks and prospects.

Notation. Upper-case bold letters are used to indicate tensors and matrices, while lower-case bold letters indicate column vectors.

2 System Description

2.1 The Mechanical System

The mechanical system analysed in this work is a car shock absorber, illustrated in **Figure 1**. Four main elements can be identified within the CAD model: the piston is a cylinder with a central hole, which allows for the oil flow during motion. This allows also absorbing the spring oscillations. The external piston radius is considered as equal to the internal piston chamber radius, without gap. The passengers and the car masses are loaded on the previously assembly via the top mounting ring.

The scheme shown in **Figure 1** represents the CAD system in functional diagram form: a spring-damper parallel system fixed to a support structure represents the car shock absorber submitted to the weight of the car and its passenger.

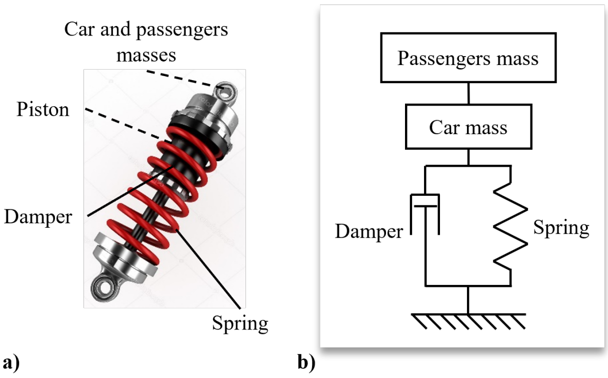


Figure 1. a) CAD model of the car shock absorber with **b)** its functional mechanical model.

The dynamic response of such system is governed by the longitudinal displacement $x(t)$ of the top chamber piston. It can be determined by solving the equilibrium equation of the system:

$$M\mathbf{a} = \sum_{i=1}^{q} \mathbf{f}_{\text{ext},i}, \tag{1}$$

where $M = \frac{m_P}{4} + \frac{m_C}{4}$ is a quarter of the overall mass of the system (i.e. car + passengers masses), $\mathbf{a}$ the acceleration, whose component along the x axis, is equal to $\ddot{x}(t)$, $\mathbf{f}_{\text{ext},i}$ the generic i-th external force applied to the spring-damper (see **Figure 2**) and q their total number.

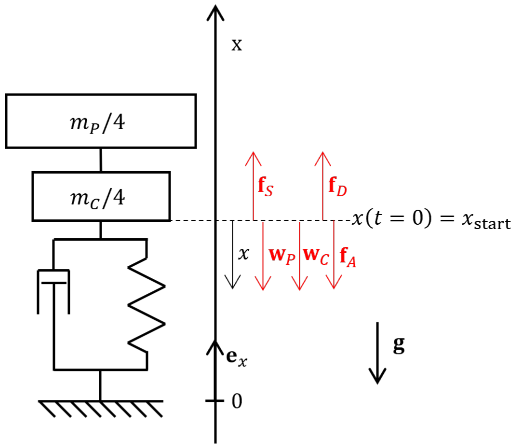


Figure 2. Functional scheme and external forces applied to a car shock absorber. The direction of the gravity acceleration is represented by the **g** vector.

The external forces applied to the spring-damper system are listed here below:

- Archimedes' force:

$$\mathbf{f}_A = -S_p h \rho_{\text{oil}} g \mathbf{e}_x, \tag{2}$$

where $S_p = \pi(r_{\text{ext}}^2 - r_{\text{int}}^2)$ is the cross-section area of the piston, r_{int} and r_{ext} are the internal and external piston radius, respectively, h is the piston height, ρ_{oil} is the oil density, g is the gravity acceleration and $\mathbf{e}_x$ the unit vector of the x axis. Note that the Archimedes' force is applied to the piston which will tend to rise in its chamber and exert a force in the direction of the vector $\mathbf{e}_x$.

- Damper force:

$$\mathbf{f}_D = -\text{sign}(\dot{x}) S_p \Delta P \mathbf{e}_x, \tag{3}$$

where ΔP is the pressure loss, computed by assuming the physical model shown in **Figure 3**. The value of ΔP is thus computed as follows (Idel'chik 1966):

$$\Delta P = \rho_{\text{oil}} \frac{\omega_0^2}{2} (\zeta_{\text{red}} + \zeta_{\text{exp}}), \tag{4}$$

where ω_0 is the oil speed inside the piston. By imposing mass flux conservation, ω_0 reads:

$$\omega_0 = \frac{\dot{x} S_p}{S_h}, \tag{5}$$

with $S_h = \pi r_{\text{int}}^2$ is the cross-section area of the piston hole. In addition, ζ_{red} and ζ_{exp} are expressed as follows:

$$\zeta_{\text{red}} = \left(\frac{1}{C} - 1\right)^2, \; \zeta_{\text{exp}} = \left(1 - \frac{S_h}{S_p}\right)^2, \tag{6}$$

with $C = 0.63 + 0.37\left(\frac{S_h}{S_p}\right)^2$.

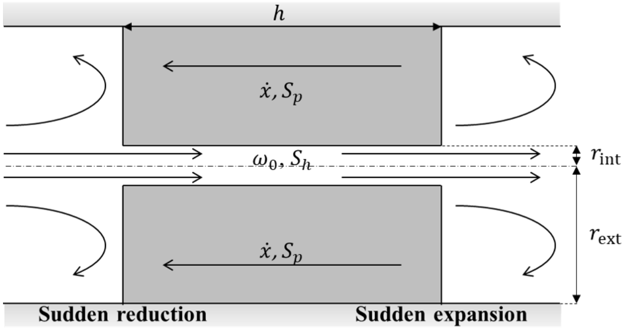


Figure 3. Different flows through the piston during its motion.

- Spring stress:

$$\mathbf{f}_S = -k(x - L_0)\mathbf{e}_x, \tag{7}$$

  where k is the spring constant and L_0 its unstreched length. Note that, at initial time, the piston position is smaller than the unstreched spring length, i.e. the system is considered as pre-loaded.

- Weight action of the passengers:

$$\mathbf{w}_P = -\frac{m_P}{4} g\mathbf{e}_x, \tag{8}$$

  where m_P is the overall passengers mass.

- Weight of the car:

$$\mathbf{w}_C = -\frac{m_C}{4} g\mathbf{e}_x, \tag{9}$$

  with m_C the overall mass of the car.

2.2 Sensitivity Analysis Based on Sobol's Indices

In this section, the fundamentals of the sensitivity analysis based on Sobol's indices (Sobol 1993) is briefly introduced. Sobol's indices allow for the identification of the input parameters having the stronger influence on the system outputs.

Consider a multiple-input-single-output (MISO) system whose transfer function is $\mathscr{M} : \boldsymbol{\xi} \in \mathbb{R}^n \longrightarrow Y \in \mathbb{R}$, where n is the number of inputs and $\mathscr{M}$ is represented by the physical model.

$\boldsymbol{\xi}^{\text{T}} = (\xi_1, \ldots, \xi_n)$ represents the vector of inputs. The variability of the inputs is modelled via random variables characterised by their probability density functions (PDFs): $\forall i \in \{1, \ldots, n\}, \xi_i \sim dP_{\xi_i}$ (which must be read: ξ_i follows a distribution of PDF dP_{ξ_i}). It is noteworthy that Sobol's indices can be defined either in the case of dependent variables or in the case of independent variables (Hoeffding 1948; Sobol 1993). In the following of this work, only independent inputs will be considered, i.e. the generic input variable cannot be expressed as a function of the remaining inputs (neither explicitly nor implicitly). This means that the probability measure of $\boldsymbol{\xi}$ is given by:

$$dP_{\boldsymbol{\xi}} = \prod_{i=1}^{n} dP_{\xi_i}. \tag{10}$$

The objective of the sensitivity analysis is to determine the relative influence of each parameter ξ_i, on the considered output value in term of variance. The Sobol's index S_i of the variable ξ_i gives the percentage of variance on the output associated solely to ξ_i. Similarly, $S_{i,j}$ gives the percentage of the output variance associated to the couple (ξ_i, ξ_j) (and this concept must be extended to all possible $2^n + 1$ combinations of input variables). Mathematically, the Sobol's index of order 1 (S_i) and of order 2 ($S_{i,j}$) can be expressed as follows:

$$\begin{aligned} S_i &= \frac{\mathscr{V}_i}{\mathscr{V}}, \\ S_{i,j} &= \frac{\mathscr{V}_{i,j}}{\mathscr{V}}, \end{aligned} \tag{11}$$

with $\mathscr{V}_i$ the variance associated to ξ_i, $\mathscr{V}_{i,j}$ the variance associated to the couple (ξ_i, ξ_j) and $\mathscr{V}$ the global variance associated to the observed variable Y of the model $\mathscr{M}$. The above quantities are defined as follows:

$$\mathscr{V} = \text{Var}(\text{E}(Y)) \tag{12}$$

$$\mathscr{V}_i = \text{Var}(\text{E}(Y|\xi_i)) - \text{Var}(\text{E}(Y)) \tag{13}$$

$$\begin{aligned} \mathscr{V}_{i,j} =& \text{Var}(\text{E}(Y|\xi_i, \xi_j)) - \text{Var}(\text{E}(Y|\xi_i)) \\ & - \text{Var}(\text{E}(Y|\xi_j)) + \text{Var}(\text{E}(Y)) \end{aligned} \tag{14}$$

where $\text{E}(Y)$ is the expected value of Y, $\text{E}(Y|\xi_i)$, the conditional expected value of Y regarding ξ_i and $\text{E}(Y|\xi_i, \xi_j)$ the conditional expected value of Y regarding the pair of variables (ξ_i, ξ_j). For a deeper insight in the matter, the reader is addressed to (Philippe and Viano 2010).

The $2^n + 1$ Sobol indices can provide precious information for the sensitivity analysis, but their computation can be prohibitive when a large number of variables are considered. In (Saltelli 2002), the author presents a numerical integration scheme requiring $N(n+2)$ simulations, with N the total number of samples allowing the computation of the n elementary indices $(S_i)_{i \in [1,n]}$ together with the n total indices defined by:

$$\forall i \in \{1, \ldots, n\}, \; \text{ST}_i = S_i + S_{i,j} + \ldots + S_{i,\ldots,n}. \tag{15}$$

The total Sobol's indices give an indication of the influence of the variable ξ_i on the considered output as an isolated variable and when ξ_i is combined with every other set of variables. For example, if $\mathrm{ST}_i \approx 0$, the variable ξ_i does not influence at all the considered output. Note that in the rest of the paper, the indices S_i are indicated as SE_i and called elementary Sobol's indices. For a deeper insight in the matter the interested reader is addressed to (Héliot 2017-2018).

3 Numerical Model of the Car Shock Absorber

The mechanical model of the car shock absorber considered in this study is built through the Modelica programming language (Fritzson 2014) with the MapleSim software (Maplesoft 2014) and interfaced with the Nebraska code wherein sensitivity analysis by means of Sobol's indices is performed.

3.1 Modelica Model

To simulate the behaviour of the car shock absorber, the Modelica programming language has been employed, and, particularly, its capability to easily export model in FMU format.

The Modelica model is shown in **Figure 4**. The Modelica Standard Library (MSL) has been used to generate the majority of the components of the mechanical system. However, the mechanical behaviour of the damper has been modelled via a customized damper force element to reproduce the constitutive law of Eq. (3). It can be noticed that the element M_C of Figure 4 (which represents a massless element with limited motion) has been introduced to limit the motion in the piston chamber (to ensure geometric/physical consistency). The weight action of the car is represented by the W_C element, whilst the one of the passengers is represented by the W_P element. The whole system is described by means of 11 parameters among which only six are considered for the sensitivity analysis.

When the car is loaded, its shock absorbers have the function to limit the spring oscillations and to dissipate them in a short time, at a given position. As a consequence, the output considered for the sensitivity analysis is the piston position at a given time $x(t)$. As a reference, the time constant for shock absorption has been set as $\tau_{\mathrm{abs}} = 2$ s with a stable piston position at $x = 0.64$ m. The reference motion parameters are given in **Table 1** and correspond approximately to a physically admissible situation. Note that the piston initial position is defined at $x_{\mathrm{start}} = L_{\mathrm{bar}} + L_{\mathrm{cham}}$ with L_{bar} the bag length in contact with the floor (assimilable to the wheel) and L_{cham} the length of the piston chamber which limit the piston displacement. At $t = 0$ s, the spring is preloaded because $x_{\mathrm{start}} < (L_0 + L_{\mathrm{bar}})$.

Once the Modelica model is created and validated, the FMU file is generated in co-simulation mode. The FMU file of the mechanical system can be represented as a black box for which input parameters with their respective intervals of definition must be defined as reported in **Table 2**. These intervals, have been chosen according to admissible mechanical conditions of the shock absorbers. In particular, the interval of definition of the parameter r_{int} is selected in order to ensure a complete or semi-periodic absorbing behaviour.

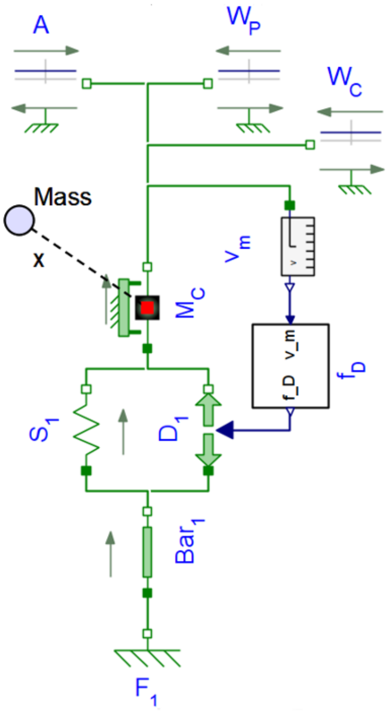


Figure 4. Modelica model of the car shock absorber.

Table 1. List of reference model parameters and their type.

Parameter	*Reference value*	*Variable*
k	9 000 N.m^{-1}	Yes
L_0	0.45 m	Yes
r_{int}	0.002 m	Yes
r_{ext}	0.03 m	No
m_P	150 kg	Yes
h	0.02 m	Yes
ρ_{oil}	884 kg.m^{-3}	No
g	9.81 m.s^{-2}	No
L_{bar}	0.5 m	No
L_{cham}	0.25 m	No
m_C	1 000 kg	Yes

3.2 Preliminary Analysis of the System

Before proceeding with the sensitivity analysis, the Modelica model has been simplified. By evaluating the orders of magnitude of the external forces, the Archimedes' force can be neglected with respect to the spring force since its value differ by three orders of magnitude. As a conse-

Table 2. List of the model input variables and the related interval.

Variable	*Interval*
k	$[8\,000; 11\,000]$ N.m^{-1}
L_0	$[0.4; 0.6]$ m
r_{int}	$[0.001; 0.005]$ m
h	$[0.01; 0.04]$ m
m_P	$[60; 360]$ kg
m_C	$[900; 1\,300]$ kg

quence, the parameter h can be removed from the input variables to be considered in the sensitivity analysis.

By applying this simplification, one can notice that, as expected, no significant variation occurs in the piston motion. The relative difference between the piston motion of the complete Modelica model and the counterpart of the simplified model vs. the time is illustrated in **Figure 5**. This difference is about 10^{-5} m, which represent 0.008 % of the stabilised motion reference (at $x = 0.64$ m), thus the two models can be considered as equivalent.

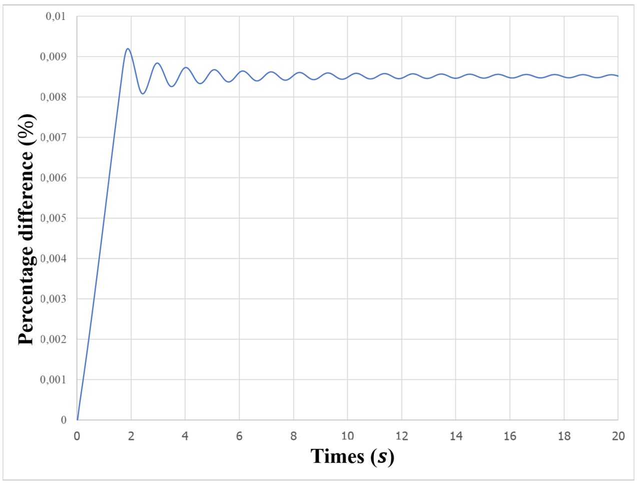


Figure 5. Percentage difference between the piston motion of the complete Modelica model and the counterpart of the simplified model in % vs. the time.

4 Numerical Results

4.1 Observed behaviours

To evaluate Sobol's indices, input variables are randomly combined into several sets. **Figure 6** shows the displacement time history of the piston for nine sets obtained by combining different values of the input variables. As it can be inferred from these results, semi-periodic (set 2, 3 and 4) or completely damped motions (set 5, 6, 7, 8 and 9) can be obtained by acting on the input variables. In extreme cases, the displacement of the piston is blocked due to a significant value of spring pre-loading (set 1). The displacement time history of the piston obtained with the reference set of parameters of **Table 1** is represented by the red curve in **Figure 6**. The values of the input variables for the other sets are given in **Appendix A**: for each set, the constant parameters are the same as the reference set (see Table 1).

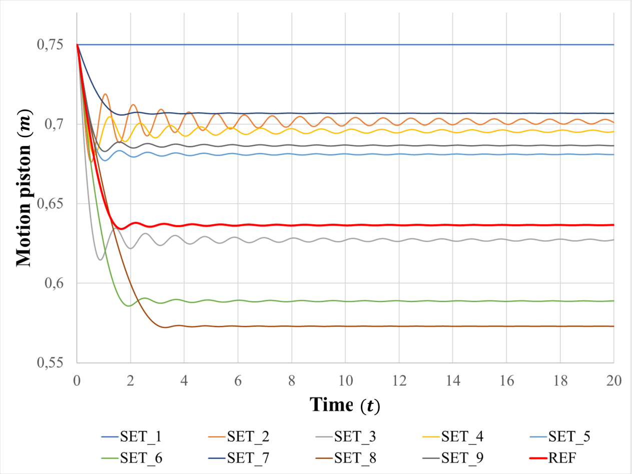


Figure 6. Piston position vs. time for 10 different sets. The red curve is the reference solution.

4.2 Sensitivity Analysis

The values of Sobol's indices, have been computed for the considered variables vs. the samples number N for two characteristic time values: $t = \tau_{\text{abs}}$ and $t = 10\tau_{\text{abs}}$.

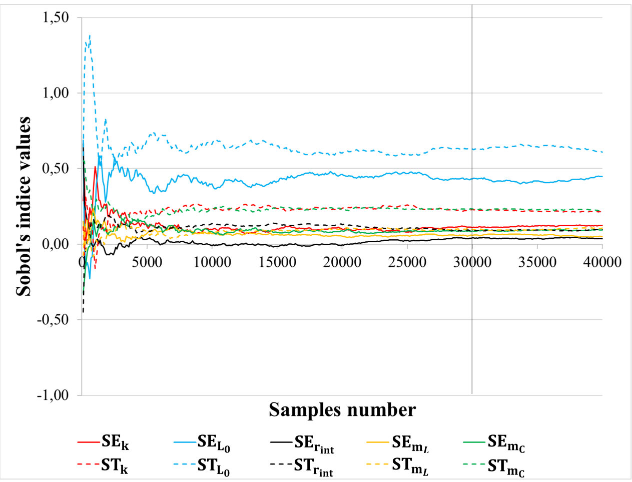


Figure 7. Sobol's indices vs. samples number N for $t = \tau_{\text{abs}}$.

The results of a Sobol's analysis can be considered reliable only if convergence is achieved. The results of **Figure 7** and of **Figure 8** highlight that for sample numbers lower than 15 000, significant fluctuations exist. These are due to numerical errors, especially for $N < 500$ where inconsistent negative values are obtained. When $3\,000 \leq N \leq 15\,000$, another inconsistency is observed since SE_{L_0} and ST_{L_0} are really not constant. For values of samples number greater than 30 000 convergence is achieved for all curves.

As a matter of fact, the standard uncertainty, $\bar{u}$, for the Sobol's indices computed for all the input variables when

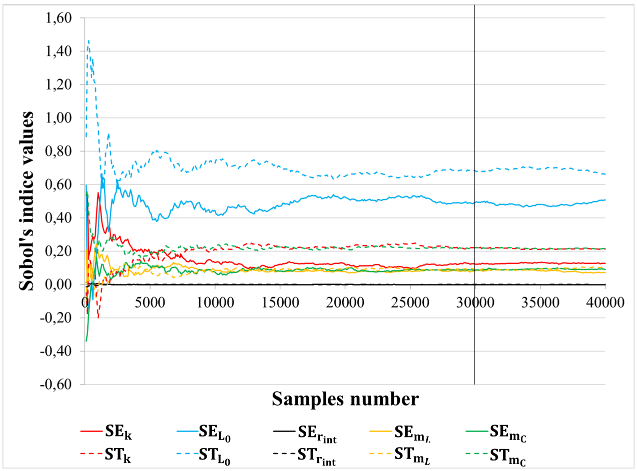


Figure 8. Sobol's indices vs. samples number N for $t = 10\tau_{\text{abs}}$.

the samples number is greater than 30000 is equal to:

$$\bar{u} = \frac{\bar{\sigma}}{\sqrt{D}} = \frac{5.78 \cdot 10^{-3}}{\sqrt{40000 - 30000}} = 5.78 \cdot 10^{-5}, \qquad (16)$$

where $\bar{\sigma}$ is the average standard deviation and D the studied range.

Table 3 reports the values of the converged Sobol's indices at $N = 40000$. It must be noticed that, in this table, all values have been multiplied by 100 to enable a better reading.

Table 3. Elementary and total Sobol's indices for $t = \tau_{\text{abs}}$ s and $10\tau_{\text{abs}}$ s in percentage. Note that, by considering their definition in section 2.2, the sum of total indices is not equal to 100 %.

Total Sobol's index	*Value at $t = \tau_{\text{abs}}$*	*Value at $t = 10\tau_{\text{abs}}$*
ST K	21.40 %	21.38 %
ST L_0	60.92 %	66.24 %
ST r_{int}	9.56 %	0.01 %
ST m_P	10.65 %	10.29 %
ST m_C	22.66 %	21.62 %
Elementary Sobol's index	*Value at $t = \tau_{\text{abs}}$*	*Value at $t = 10\tau_{\text{abs}}$*
SE K	12.39 %	12.77 %
SE L_0	44.93 %	50.85 %
SE r_{int}	3.70 %	0.00 %
SE m_P	4.93 %	7.16 %
SE m_C	9.32 %	9.15 %
Interactions	24.73 %	20.07 %

A total Sobol's index can help understanding the influence of a given input variable on the piston motion when combined with other variables. For instance, in the case of the total index of m_P at $t = \tau_{\text{abs}}$ s, its value is 10.65 %, which is at least two times smaller than the other ones, like r_{int}.

As far as variable L_0 is concerned, always at $t = \tau_{\text{abs}}$ s, a significant difference exists between its elementary Sobol's index, 44.93 %, and the total one, 60.92 %. This means that this parameter plays a crucial role on the piston motion in combination with other parameters.

Finally, by prioritising the other indices, it can be concluded that the parameter L_0 is the one influencing the most the motion of the piston.

The same remarks can be repeated for $t = 10\tau_{\text{abs}}$, except for r_{int} whose elementary and total Sobol's indices become smaller than the ones related to the other input variables. This is due to the fact that r_{int} is involved only in the expression of the damper force. This force being function of the piston speed, it is possible to link indirectly r_{int} with the others parameters. Thus, when the system reaches a stable state, the piston velocity goes to zero and the influence of parameter r_{int} too.

The results of **Table 3** also highlight that the parameters k, m_P and m_C present similar values of the total Sobol's index at both time, $t = \tau_{\text{abs}}$ s and $t = 10\tau_{\text{abs}}$ s. Nonetheless, if a greater range of r_{int} is considered, the influence of this parameter on the motion regime can be observed. Indeed, for values of r_{int} greater than 0.005 m, the motion of the piston is mainly semi-periodic or completely periodic without stable motion. This highlights the importance of correctly defining the interval of the input variables to be used during the sensitivity analysis. Furthermore, inasmuch as r_{int} is the input variable with the weakest influence on the piston motion, its value can be set randomly without influencing the final result. Conversely, particular care must be put in choosing the value and the interval of definition of the most influencing variables (like L_0).

The coupling of the FMU file of the car shock absorber Modelica model with the Nebraska code allows carrying out for fast simulations to be carried out. In fact, the Modelica model, shown in **Figure 4**, requires approximately 4.50 s to run while the sensitivity analysis with Nebraska (which makes use of the FMU file generated from MapleSim) takes approximately 5000 s for a total of 280000 simulations. All the simulation presented in this paper have been carried out on a DELL® laptop with 32 Gb RAM and an Intel core i7-10610 processor. From the analysis of the time required to execute each step of the simulation, one can conclude that about 50 % of the overall time is required to solve (and integrate) the problem, while the remaining 50 % of the time is needed for post-processing of results. In particular, the time required to execute each step is:

1. Equations generation (1.131 s);
2. Equations handling and manipulation (0.219 s);
3. Initial values computation (0.224 s);
4. Integration step (0.710 s);
5. Post processing of results (2.216 s).

5 Conclusion

In this work a sensitivity analysis of a car shock absorber model, carried out through the coupling of an in-house code (Nebraska) based on Sobol's indices and a FMU file generated via MapleSim, has been presented.

The goal is to identify a reduced model capable of describing the motion of the piston. To this purpose, the influence of the different physical phenomena on the considered output has been investigated. The system, described via several geometrical and mechanical parameters, has been firstly analysed by observing the order of magnitude of the different forces acting on the system. As a first result, the initial model has been simplified by removing those forces whose mechanical effects are negligible when compared to others.

The simplified model was then used to generate the FMU file, which was coupled with the based-Fortran code Nebraska through a Python code (in-house developed) which carried out the sensitivity analysis via Sobol's indices.

The analysis of both the elementary and total Sobol's indices of the input parameters of the mechanical system allowed determining that the spring unstreched length (L_0) plays a crucial roles in the piston motion while the influence of r_{int} is negligible. Note that this result holds solely for the output considered in this study, i.e. the position of the piston when a stable state is reached.

This study represents a first application of FMU to simulate mechanical systems with the aim of performing sensitivity analysis. A further extension of the approach presented in this work could be its application to more complex systems. In this context, the more influential subsystems, determined as a result of a sensitivity analysis based on Sobol's indices, or subsystems involving complex physical phenomena that must be accurately reproduced, could be replaced by a external model imported via a FMU file. As far as prospects of this work are concerned, the FMU can be generated from dedicated finite element models or meta-models, integrating the relevant physical responses. This can help reducing the computational cost of the sub-system simulation required within the global model of the mechanical system developed in MapleSim.

Acknowledgements

The authors would like to thank Laurent JACQUI who developed the Nebraska code.

References

Chastaing, G., F. Gamboa, and C. Prieur (2015). "Generalized Sobol sensitivity indices for dependent variables: numerical methods". In: *Journal of Statistical Computation and Simulation* 85.7, pp. 1306–1333. DOI: 10.1080/00949655.2014.960415.

Fritzson, Peter (2014). *Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach.* ISBN: 9781118989166. DOI: 10.1002/9781118989166.

Héliot, Maxime (2017-2018). "Studies and development of a sensitivity analysis tool and application ti an aerothermal model". MA thesis. Cranfiel University.

Hoeffding, Wassily (1948). "A Class of Statistics with Asymptotically Normal Distribution". In: *Annals of Mathematical Statistics* 19.3, pp. 293–325.

Idel'chik, I.E. (1966). *Handbook of hydraulic resistance*. Ed. by IPST Staff D. Grunaer P.E. The U.S. Atomic Energy Commission and The National Science Foundation.

Janon, Alexandre (2019-06). "Global optimization using Sobol indices". working paper or preprint. URL: https://hal.archives-ouvertes.fr/hal-02154121.

Maplesoft, Copyright (2014). *MapleSim User's Guide*. ISBN: 9781926902326.

Marrel, Amandine et al. (2009). "Calculations of Sobol indices for the Gaussian process metamodel". In: *Reliability Engineering and System Safety* 94.3, pp. 742–751.

Philippe, Anne and Marie-Claude Viano (2010). "Cours de probabilités : Modèles et Applications".

Plogert, Klaus (1996). "The tailoring process in the German V-Model". In: *Journal of Systems Architecture* 42, pp. 601–609.

Saltelli, Andrea (2002). "Making best use of model evaluation to compute sensitivity indices". In: *Computer Physics Communications* 145.(2), pp. 280–297.

Sobol, I.M. (1993). "Sensitivity Estimates for Nonlinear Mathemitical Models". In: *MMCE* 1.4, pp. 407–414.

Acronyms

MBSE	Model-Based System Engineering
MISO	Multiple-Input-Single-Output
SE	Sobol Elementary Index
ST	Sobol Total Index
FMU	Functional Mock-up Units
PDF	Probability Density Function
CAD	Computer Aided Design
MSL	Modelica Standard Library

Glossary

N	Input variables sets number	–
d	Number of input variables	–
ζ	Proportional factor in singularity pressure loss	–
τ_{abs}	Absorbing motion time	s
ΔP	Pressure loss	Pa
k	Spring constant	N.m^{-1}
L_0	Spring unstreched length	m
r_{int}	Internal piston radius	m
r_{ext}	External piston radius	m
h	Piston thickness	m
x_{start}	Piston initial position	m
L_{cham}	Length of the piston chamber	m
L_{bar}	Length of the bar linked to the floor	m
ω_0	Fluid speed	m.s^{-1}
g	Gravity acceleration	m.s^{-2}
S_p	Piston surface	m^2
S_h	Piston hole surface	m^2
m_P	Passengers mass	kg
m_C	Car and passengers mass	kg
ρ_{oil}	Oil density	kg.m^{-3}
$\bar{u}$	Mean of standard uncertainly	–
$\bar{\sigma}$	Mean of the average standard deviation	–

Appendix A

Table 4. Variable parameters data for ten different sets.

Parameter	*set 1*	*set 2*	*set 3*	*set 4*	*set 5*	*set 6*	*set 7*	*set 8*	*set 9*	*ref*
K N.m^{-1}	10768	10935	9513	9811	9830	9329	8951	9306	10345	9000
L_0 m	0.585	0.498	0.475	0.562	0.480	0.479	0.561	0.407	0.533	0.450
r_{int} (10^{-3} m)	3.63	4.20	3.24	3.37	2.49	2.25	1.77	1.75	2.42	2.25
m_P kg	239	140	178	213	111	196	135	359	310	150
m_C kg	964	1182	1172	1252	1086	1287	1156	908	1151	1000

Detailed White-Box Non-Linear Model Predictive Control for Scalable Building HVAC Control

Filip Jorissen[1] Damien Picard[1] Kristoff Six[1] Lieve Helsen[1,2]

[1]Mechanical engineering, KU Leuven, Belgium filip.jorissen@kuleuven.be
[2]EnergyVille, Genk, Belgium

Abstract

Grey-box and black-box MPC approaches for building HVAC applications often use lumped, low-order models with a low level of detail. While such models require smaller computation times, their accuracy is limited and there are practical constraints related to data collection, how to deal with multi-zone buildings and they often do not explicitly model the building HVAC equipment. In this paper we present an alternative approach based on detailed white-box models. TACO, a custom toolchain that builds upon physics-based Modelica models and JModelica, is used to efficiently solve the resulting optimisation problems. This paper presents a realistic case study model of 79 zones and OCP results for this case study are discussed, demonstrating the feasibility of the approach and the underestimated potential of detailed white-box MPC.

Keywords: Optimal control of hybrid systems, HVAC, white-box modelling, building automation, TACO, JModelica, MPC

1 Introduction

Building Heating, Ventilation and Air Conditioning (HVAC) accounts for 15 % of the world final energy use (International Energy Agency 2019). While building design standards become stricter, the building energy use is to a large extent determined during operation, when control and the available building flexibility play an important role. Model Predictive Control (MPC) is a methodology for controlling the building HVAC equipment during this operational phase. The goal of MPC is typically to find the HVAC control set points (or control actions) that lead to the lowest (operational, environmental or other) cost while ensuring that comfort and other constraints are respected.

Most MPC research and companies use data-driven approaches such as grey-box and black-box modelling. These approaches fit model parameters using measurements from a real building and thus rely on the availability of qualitative data, which may not be easy to obtain in practice. Moreover, the main challenge of grey-box modelling still is the need for a robust parameter estimation method (Drgoňa et al. 2020). For data-driven approaches, building zones are therefore often lumped to limit the complexity during the training phase. This lumping phase inevitably leads to a loss of detail, which could be problematic when different parts of the lumped zones have a different behaviour. E.g. the set points for a lumped zone with average heat load may not be sufficient to heat the two consisting zones that have low and high heat load respectively. Perhaps this infrequently poses problems in practice, but managing these kinds of problems takes time (which is expensive and a liability in a commercial context), and model simplifications are likely to cause reduced energy efficiency and thermal comfort throughout the building lifetime. Furthermore, simplified models typically require the development of a subcontroller that maps MPC set points to device set points, which can be cumbersome (Drgoňa et al. 2020).

Using models with a higher level of detail leads to additional advantages. For instance, the detailed physics-based model can be used to benchmark the actual system, it can be used for fault detection (Frank et al. 2016), model and results are easier to interpret ('Explainable AI'), the model could be easier to adjust and update since it is physically interpretable, it could be reused for retrofit analyses, or even to visualize the building using augmented reality. Sometimes white-box models are used to train a grey or black-box model. Why would you train a simplified model if you already have a detailed model? Often the answer to that question relates to computation time, or to the inability of the white-box tool to perform optimizations altogether. Our goal is to improve the solver, rather than to simplify the model. We start from physics-based Modelica models, and JModelica (Åkesson et al. 2010) to keep the original model accuracy, at acceptable computational cost. For a recent overview of MPC for building applications we refer to (Drgoňa et al. 2020).

Considering the above and also the economic context we present an MPC development workflow that is designed with scalability in mind. More specifically, it is designed to be robust against modelling errors (user error), fast to use, easy to maintain and to extend, generic for many types of HVAC devices and HVAC schematics, low-demanding with respect to expertise to implement and operate, and systematically applicable to (a class of) buildings. Our approach uses detailed white-box Modelica models as a starting point, which can be refined using measurement data during operation. The use of Modelica

DOI
10.3384/ecp21181315

unlocks efficient numerical algorithms that scale well with the problem size and it facilitates collaborations on model development (e.g. within IBPSA project 1). An often quoted disadvantage of white-box modelling is the effort required to describe building properties (Drgoňa et al. 2020). By using automated modelling workflows, this implementation effort is strongly reduced.

Few researchers have demonstrated white-box optimal control methodologies that scale to the size and complexity that is required for large buildings, assuming that the goal is a qualitative implementation that sets individual set points for individual actuators. (Sturzenegger et al. 2016) have presented a respectable physics-based modelling approach, BRCM, that uses bi-linear models. A 20-zone model with 300 states has been demonstrated and model reduction has been used to reduce the number of states to 55. While the total conditioned floor area was 6000 m^2, only one of the six floors (1000 m^2) was actually modelled. In this work we present a white-box modelling approach that has been applied to case-study model that has 3421 state variables, 217 control inputs and 79 zones that span 10 000 m^2. Furthermore, multiple *non-linear* AHUs and other HVAC equipment are included instead of only using bi-linear HVAC. Th resulting proof-of-concept Optimal Control Problem (OCP) demonstrates the feasibility of this approach even for complex multi-zone buildings. Furthermore, the presented optimisation results illustrate the potential of our approach. We present OCP results in this work instead of MPC results since results of a single optimisation with a long horizon are easier to interpret than a concatenation of the first intervals of a sequence of optimisations. We do not consider uncertainties on disturbance forecasts or modelling errors in this work. Clearly, our approach would have to deal with these uncertainties, as would any other approach. Note that the model can indeed be used for MPC too, since MPC is controlling the modelled building (located in Luxemburg) since a few months. A preliminary interpretation of the results suggests that forecasting errors of weather and occupancy dominate modelling errors.

2 Methodology

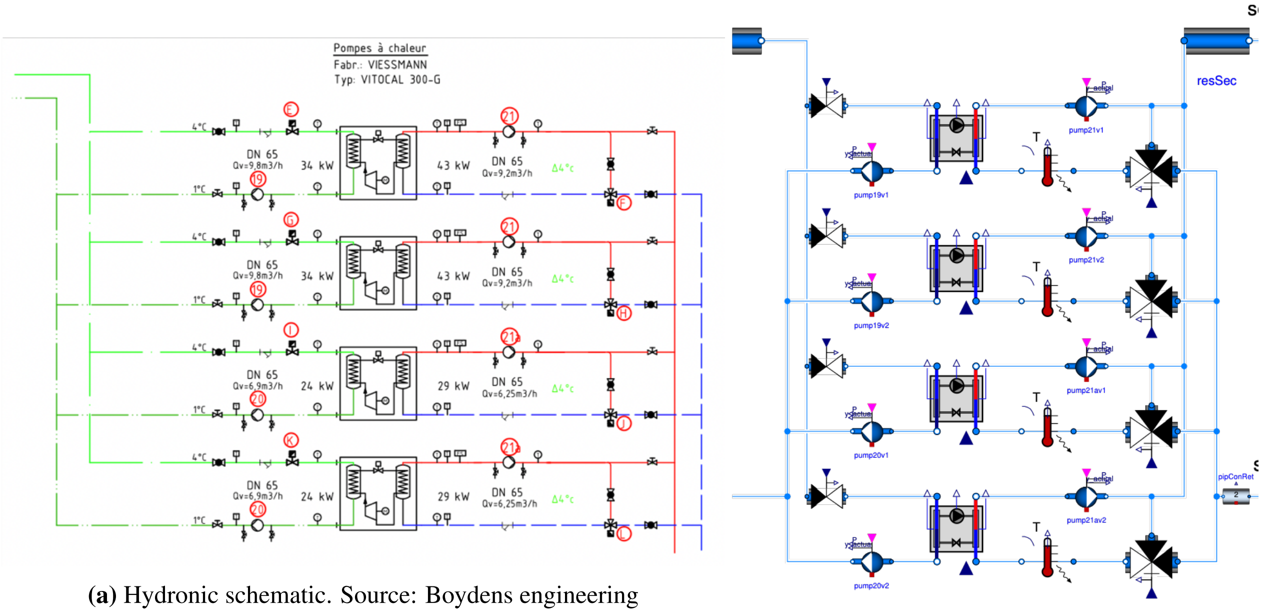
Our methodology uses detailed, physics-based (white-box) models. We start by mapping each physical component to a respective model. For the building envelope, zones are grouped such that rooms that can be controlled individually, are also modelled individually. For the building HVAC, components can be mapped to models one-to-one as illustrated in Figure 1, or each group of components that serves the same function is modelled using one component model. E.g. each pump, valve, heat exchanger, etc. is modelled individually, but a set of two redundant pumps, or a set of 20 solar collectors has 1 model per set. The component models are implemented using the modelling language Modelica. For a description of the building envelope models and equations, we refer to the Modelica IDEAS library (Filip Jorissen, Reynders, et al. 2018; KU Leuven and 3E 2012) and the Modelica Buildings library (Michael Wetter et al. 2014; Wetter et al. 2019). The models are parameterised using parameters that are commonly available in the technical specifications of the building HVAC equipment. Instantiating the Modelica component models and making the required connections between them is a tedious and error-prone process. Therefore a browser-based tool has been developed that automatically generates the required Modelica/IDEAS code. This graphical user interface (GUI) only allows valid building geometries to be specified, thereby avoiding user errors. The GUI export of this geometrical information is automatically mapped into the Modelica model as illustrated in Figure 2. Building geometry information, such as orientations and surface areas, is automatically deduced from the export, while the user specifies additional information such as material layers through the GUI options.

Code generation of connections is automated, which rules out errors such as unconnected ports or ports with too many connections. Furthermore, sanity checks are performed such as identification of unconnected devices, (un)realistic thermal insulation values in outer walls, missing parameters, etc.

The resulting workflow leads to a set of interconnected Modelica component models that describe the relevant physics:

- Thermal conduction, convection and radiation within and outside of the building envelope,
- Thermal inertia of the building,
- Solar heat gains (considering glazing type and shading) and internal heat gains from occupants,
- Pressure-driven flow rates for aerolic, hydronic system, including non-linear valve and damper models,
- Pumps and fan powers,
- Temperature dependent and mass flow rate dependent heat flow rates in emission systems,
- Temperature dependent and flow rate dependent efficiencies in production and distribution systems.

The resulting model is non-linear. The most important non-linearities are the relation between flow rate and pressure in fluid flow networks, and the strongly non-linear relation between fan/pump power and flow rate. For more details about these models see (Filip Jorissen, Reynders, et al. 2018; F. Jorissen, Boydens, and L. Helsen 2019; Filip Jorissen, Michael Wetter, and Lieve Helsen 2018; F. Jorissen, Boydens, and L. Helsen 2017; Filip Jorissen 2018).



(a) Hydronic schematic. Source: Boydens engineering

(b) Mapped model

Figure 1. Illustration of one-to-one component to model mapping

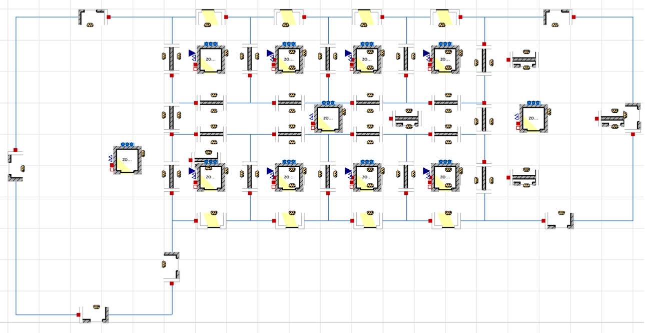


Figure 2. Automatically generated Modelica model. The drawn icons depict zones, windows, exterior walls and interior walls. Note that window icons are drawn on top of wall icons.

That Modelica model is translated into an optimisation code using our Toolchain for Automated Control and Optimisation (TACO) (F. Jorissen, Boydens, and L. Helsen 2018), which extends the JModelica framework (Åkesson et al. 2010). TACO identifies what equations must be solved for what variables, splits variables that depend on optimisation variables from those that do not, and performs preprocessing on linear state[1] interdependencies to speed up code evaluation. The continuous time problem is discretised at a user-defined non-equidistant set of points in time. CasADi (Andersson et al. 2019) computes the equation derivatives and generates C-code for evaluating the objective, constraints, derivatives and other outputs. The compiled code is coupled to a gradient-based NLP solver and is portable to other (linux) machines. For more details about the translation process that TACO performs we refer to (F. Jorissen, Boydens, and L. Helsen 2018).

3 Case study

In this paper we present a case study building, Solarwind, on which the presented methodology is applied. The office building has a conditioned floor surface area of 10 000 m^2 and was designed to be an examplary showcase towards Luxembourg and the European design and construction industry of a holistic and integrated operation and design sustainability approach for future oriented office buildings. It uses geothermal heat pumps, concrete core activation (CCA), solar collectors, solar PV, passive cooling, indirect evaporative cooling, a pellet boiler, a large storage tank, etc. This building has been described in detail in Chapter 2 of (Filip Jorissen 2018). Chapter 10 describes an MPC for (a part of) the same building. That model is substantially smaller and less detailed and complex than the OCP that we present here. It also required manual building-specific simplifications in the HVAC models that conflict with the 1-to-1 mapping philosphy and the expertise requirements that were outlined above. We now explain the building and its model together with the main differences from the earlier MPC implementation.

3.1 Building envelope

The new model consists of six floors (instead of four) that are modelled using 79 zones (instead of 32). We assume that 6 people are present in each zone, between 7:00 and 19:00 on week days. Three zones have 9 occupants instead of 6, to show the influence on the CO$_2$ concentration in the result section. Solar shading is not modelled to artificially increase the heat load and to make the optimisation problem more challenging to solve. A weather data

[1] We distinguish state variables that are computed by a differential equation from algebraic variables that are computed from an algebraic equation.

file for Uccle, Belgium is used. The building uses triple glazing and is strongly insulated with a U-value of about 0.1 W/m^{2}K.

3.2 Emission system

Ventilation is provided using six air handling units (instead of 2), for which the supply and extraction fan pressure, the heating coil valve, humidifier, heat recovery bypass (2 dampers), indirect evaporative heat exchanger control signal and active chiller control signal are optimised. Most zones have a supply Variable Air Volume (VAV) and an extraction VAV for which one control signal is optimised. A VAV control signal of 0 % corresponds to a *set point* of 50 % of the nominal flow rate[2]. Furthermore, each VAV has a heating coil for which the heating fluid flow rate is optimised using a two-way valve. The supply water temperature of the VAV coil collector is controlled using a three-way valve.

The concrete ceilings are heated or cooled using CCA, except for the top floor, which uses chilled ceilings (CC) instead. Each floor is subdivided in about six CCA or CC sections. Each section spans one or more zones, is connected to one of three collector connections (south, north, or top floor), and its total flow rate is controlled using one two-way valve. The supply temperature of each connection (i.e. group of sections) is controlled at the collector using a three-way valve.

3.3 Production system

The main collector draws water from either a geothermal heat pump, or the geothermal borefield using a series of pumps and heat exchangers (see Figure 2.3 in (Filip Jorissen 2018) for more details).

A pellet boiler and solar collector feed hot water in a 20 m^3 storage tank, which is used by the AHU heating coils. The corresponding fluid loop consists of 2 heat exchangers, 6 pumps, 3 three-way valves and a pressure-independent valve. Three domestic hot water tanks that are present in the building are not modelled for this study since realistic load profiles are not available.

3.4 Objective and constraints

We minimize the electrical power use and the pellet boiler thermal power, the latter being scaled by 1/3 to consider that pellet fuel has a different price than electricity. Various constraints are enforced, among them the minimum and maximum supply air temperature (16 °C - 26 °C), (building owner specified) zone temperature limits (22 °C - 24 °C) and a CO_2 concentration upper limit of 1000 ppm.

3.5 Horizon

For the purpose of this paper, we discretise the model in 720 intervals of 1 hour (i.e. 1 month). The resulting OCP is instantiated 12 times to optimise a full year. We use an OCP since it simplifies interpretation of the results compared to an MPC running in a receding horizon fashion. Each OCP has the same initial state. For easier result interpretation we assume that all HVAC equipment is enabled 24/7.

4 Results

The resulting model has 217 control inputs, 3421 state variables and 41 577 algebraic variables as reported by the Modelica simulation software Dymola 2020. Since the OCP has 720 control intervals, 156 240 control inputs are optimised in each OCP. Computation time for one OCP (single-core, 2 GHz) is a few days, depending on the chosen convergence tolerance. Note that an MPC using this model is orders of magnitude faster since its horizon is shorter and warm starting can be used. At the time of writing, a revised version of this model is successfully controlling the modelled building, where the MPC is updated every 15 minutes. Computation speed is thus not a problem, even for white-box models of this size.

Figures 3 - 6 present OCP results for January, April and August. We use these figures to illustrate some of the strengths of our detailed white-box MPC approach, without discussing each sub-plot in detail. Despite that these are in fact OCP results instead of MPC results, we consider that the results are representative for MPC. The presented results are direct outputs of the OCP. This is a first strength: since the model is detailed, the outputs are detailed too, which allows a thorough analysis of the results without requiring additional simulations to see the impact on subsystems that are lumped in the optimisation.

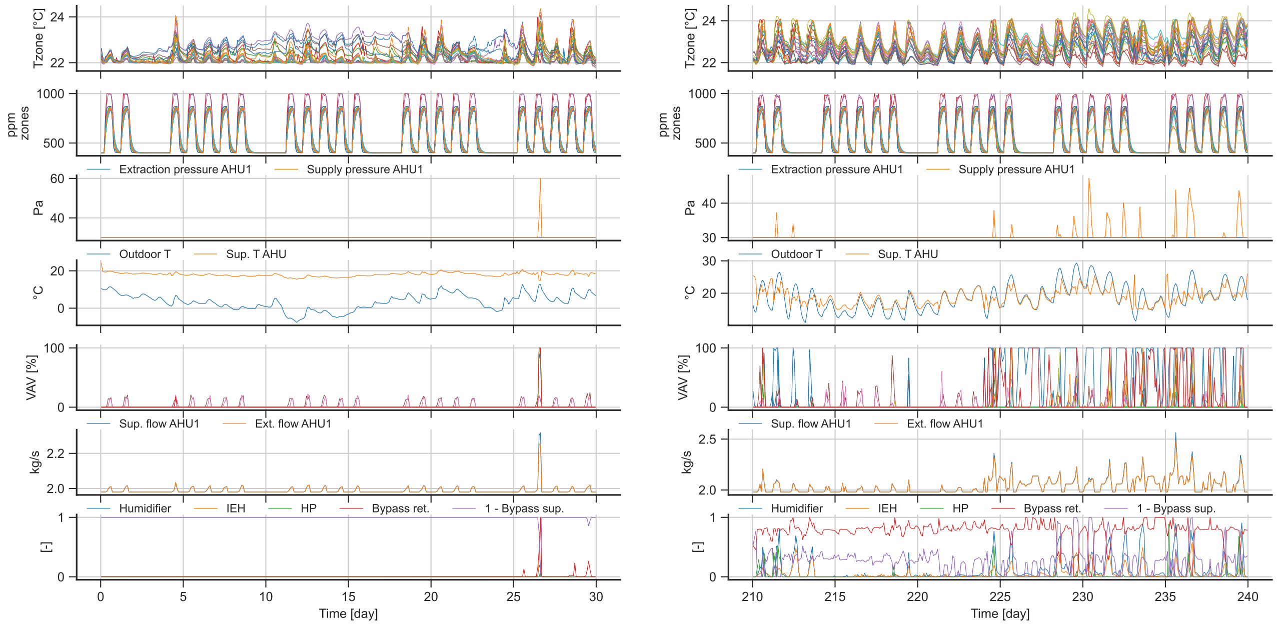
4.1 System coordination and constraints

In the top sub-plot of each figure, relevant zone temperatures are indicated, while CO_2 constraints are also indicated in the top of Figure 3. The results show that the operational constraints of 22 °C - 24 °C are respected, as well as the upper limit of 1000 ppm CO_2. At the same time, the third sub-plot in Figure 3 indicates that the AHU fans usually operate at low pressures of 30 Pa while the nominal system pressure is around 300 Pa. At the same time, most VAVs are at or around the minimum opening of 0%. However, the VAVs that correspond to zones with a larger occupancy are occasionally opened to avoid violating the CO_2 constraint. During the summer period, the fan pressures are occasionally increased to accommodate the peak cooling load due to solar heat gains. This illustrates the second strength of MPC: the coordination of multiple devices to ensure that constraints are achieved at minimum cost.

4.2 System dynamics

A third strength of MPC is illustrated in Figure 4: its ability to anticipate heating and cooling loads and to deal with them accordingly using fast (VAV) and slow (CCA) reacting systems. During the August period, large heat gains are present since we deliberately omitted solar shading in the model. This causes both the upper and lower temperature bounds to be reached within the same day (and in

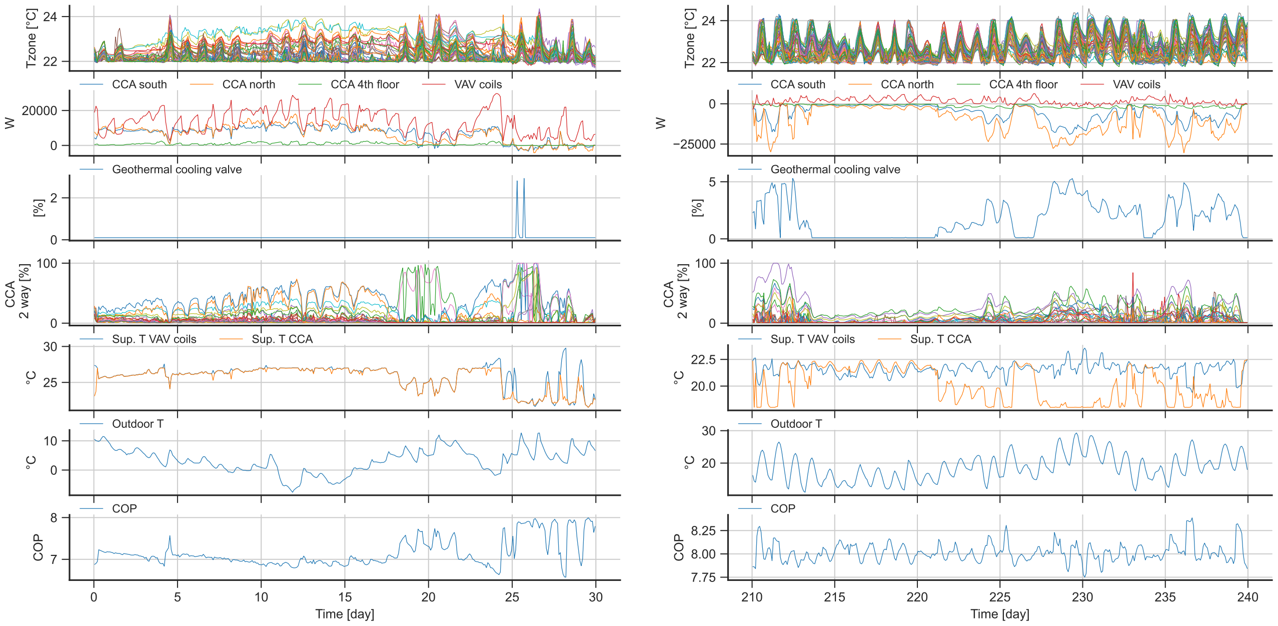
[2] The set point may not be obtained if the fan pressure is too low.



(a) January

(b) August

Figure 3. Results that are relevant to air handling unit 1



(a) January

(b) August

Figure 4. Heat pump, geothermal cooling, CCA and VAV operation.

different zones). The cooling is spread over the entire day and the cooling peak is out of phase with the temperature peaks, see sub-plots 1 and 2. The building is thus precooled, thereby considering the emission system delay.

This is illustrated once more during the winter period around day 25, when there is a period of large solar irradiation. During the three days preceding the solar peak, the CCA thermal power is relatively low and shifted towards the VAV coils to avoid excess heat being stored in the building thermal mass, which would overheat the building during the sunny period.

4.3 Hybrid systems

A fourth strength of MPC is its ability to coordinate between multiple heating and cooling sources and to use the one with the lowest cost. This is illustrated in Figure 3. During winter, the AHU bypasses (see last sub-plot) are both closed since heat is valuable during winter. During summer, 4 AHU cooling options exist (free-cooling bypasses, humidifier[3], indirect evaporative heat exchanger (IEH), chiller (HP)), which are all used. Even when no immediate cooling is required and the outdoor temperature is low, the AHU bypasses are opened. This cooling option *reduces* the fan power since the AHU internal pressure drop decreases. Note that the lower supply air temperature limit of 16 ° C is respected. When the outdoor temperature is higher than the indoor temperature, *cold* is recovered by *closing* the bypasses. When this does not suffice, the humidifier and IEH are used. They have the lowest cost since they only require a pump to be operated. In some cases, the chiller is used, which is the most expensive cooling option. Thanks to the good coordination of all other devices, this rarely happens.

Another example is shown in Figure 5, which shows the operation of the solar collector, pellet boiler and storage tank. During winter, the sun is at a low altitude due to which the collector heat losses are often larger than the solar heat gains. The solar thermal collector valve (see subplot 5) is therefore only opened when the sun intensity is sufficiently strong to reach a positive thermal power (see sub-plot 2). The remaining high-temperature heat load is provided by the pellet boiler. During the summer period, there is abundant (free) heat available from the solar collector due to which the pellet boiler is never activated.

4.4 Operational limits

A fifth strength of MPC is its ability to operate the system at its limits. For instance, the geothermal borefield can be used to passively cool the building using the CCA. A borefield temperature of 15 °C is assumed. Directly using this low temperature would be the least costly since less mass flow rate (and thus pump power) is required to achieve the same heat flow rate. However, condensation of moist air can occur on CCA when its temperature is low. Therefore, a minimum supply water temperature of 18 °C was set. Figure 4 clearly shows that this minimum supply water temperature is used during periods of large cooling load (see sub-plot 5).

4.5 Efficient operation

A sixth strength of MPC is its ability to operate the available equipment at an efficient operating point. Figure 4 illustrates this for the heat pump operation in the three last sub-plots. For lower outdoor temperatures, larger heating powers are required. This increases the required heat pump supply water temperature, which reduces the heat pump COP (excluding pump power). Note day 4, where a day of large solar intensity (see sub-plot 2 of Figure 5) 'charges' the building, immediately reducing the heating requirements, increasing the COP and also increasing the COP during the days after the heating event, despite the decreasing outdoor temperatures.

4.6 Exploiting flexibility

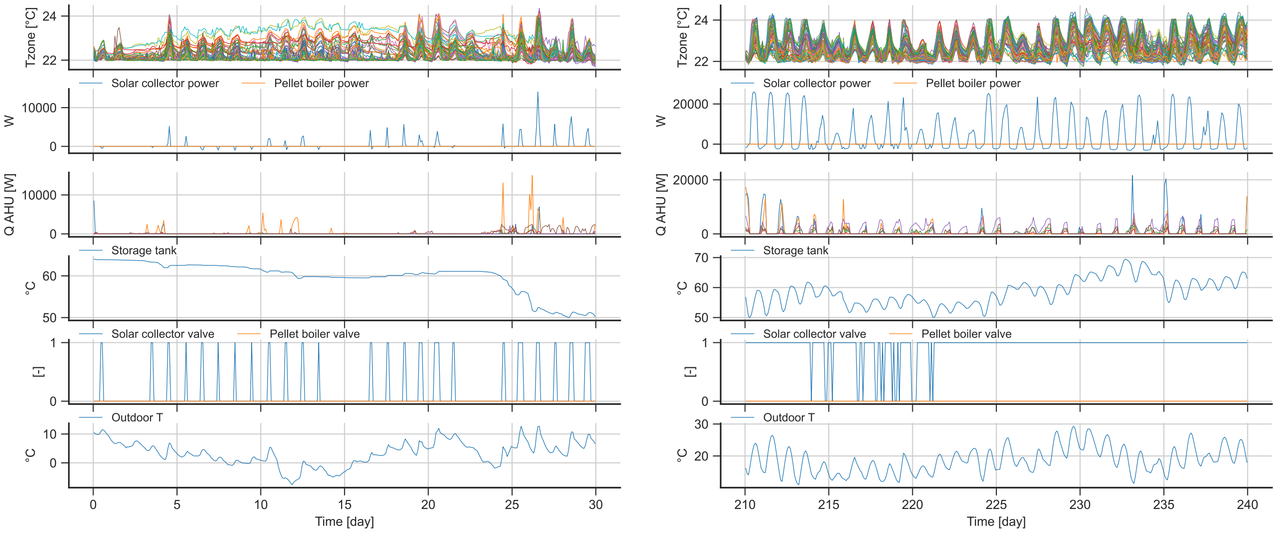
The seventh MPC strength is to shift heating loads using the available system flexibility. We already discussed precooling using the CCA. Note that this effect could have been more pronounced if the building solar heat gains were smaller or if the comfort band were larger than 2 K, allowing more drift of the indoor temperature. Hitting both the upper and the lower comfort bound within the same day limits the available system flexibility.

Additionally, the hot water storage tank flexibility is used, which is allowed to fluctuate between 50 °C and 90 °C. E.g. during the winter period in Figure 5 solar heat is accumulated for three weeks until the last days of the month when the heat is most useful. During summer, solar heat is abundantly available and is even dissipated by not closing the solar collector valve at night. Otherwise the storage tank upper temperature limit of 90 °C would be violated at day 233. Figure 6b shows the solar collector operation in April, where solar heat is stored between sunny (see sub-plot 2) days 95 and 107 and used during overcast days 108, 109, 117 and 118.

4.7 Reliable performance

Finally, we discuss the system operation during the intermediate season, when well insulated buildings are often hard to control. For instance, on cold but sunny days the building can be overheated by conventional heating curve-based controllers that increase heating set points despite substantial solar heat gains. It is hard to determine whether the building should be in heating mode, cooling mode, or perhaps even in both within the same day. It is interesting to see how MPC copes with such scenarios, which is illustrated for the month of April in Figure 6. Our MPC controller does not formally define a 'mode' but based on the second sub-plot left we conclude that MPC is in heating mode during the first two days and in cooling mode for three days. During the remainder of the month, the system is in a neutral mode despite the outdoor temperature ranges from 0 °C to 20°C. In this neutral mode,

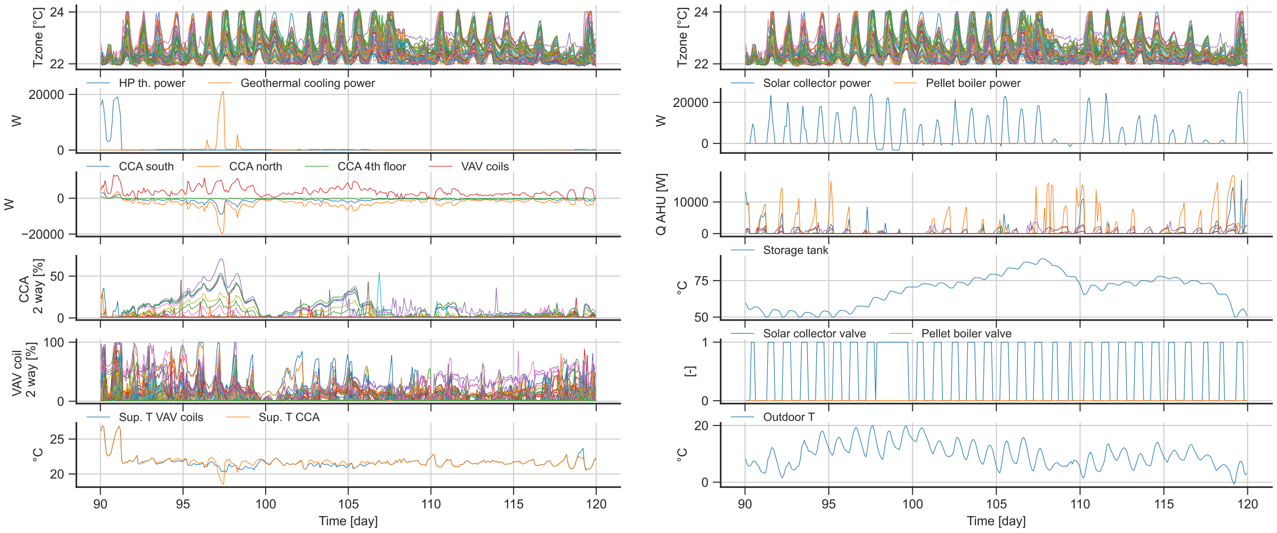
[3] The OCP does not contain a humidity constraint due to which the humidifier can be used to cool.



(a) January

(b) August

Figure 5. Solar collector and pellet boiler operation.



(a) Heat pump, geothermal cooling, CCA and VAV operation.

(b) Solar collector and pellet boiler operation.

Figure 6. System operation in April.

the system is *simultaneously heating and cooling by exchanging heat within the system* (see sub-plot 3 in Figure 6a). More specifically, the relatively warm CCA return water is recirculated through the VAV heating coils, heating the supply air. The fifth sub-plot shows that the CCA heat extraction is focussed on just a few CCA circuits and the sixth sub-plot shows that heat dissipation does not use all VAV circuits. This operating mode is particularly efficient since fewer pumps and even the heat pump need not be enabled. Perhaps more importantly, the low temperature differences allow simultaneous heating and cooling in different zones and the comfort constraints are respected. Furthermore, heat is also exchanged between warm and cold CCA circuits, but this is not indicated in the graph. Therefore, the total heat exchange is even larger than indicated. Interestingly, this operating mode is automatically discovered thanks to the level of detail of our white-box MPC.

5 Conclusion

This paper presents a detailed white-box MPC approach for buildings that is designed for commercial MPC applications, including large and complex buildings. The approach maps physical objects and devices into their respective Modelica models using a custom browser-based graphical user interface. The resulting Modelica model is translated into an efficient MPC code using TACO. The approach is applied to a case study office building of 10 000 m^2, resulting in a model of 79 zones. A discussion of OCP results for three individually optimised months shows the strengths of the toolchain. All presented strengths are automatically achieved by our white-box OCP, without substantial tuning or training. Each optimisation consists of 720 intervals of one hour, resulting in 156 240 control inputs and 29 935 440 algebraic variables, which demonstrates the feasibility of large-scale white-box optimisation. The presented case study is currently being controlled by MPC, hence also demonstrating the computational feasibility of this approach for direction optimisation. Furthermore, the presented OCP approach can be compared to other control methodologies, e.g. within the frame of IBPSA Project 1, WP 1.2 BOPTEST (Blum et al. 2019). Future work will present real-life, operational results of our MPC approach.

Acknowledgements

We thank Boydens Engineering, Schuler SàRL, DRC Technology Sa, Menerga nv and Climalux Sa for enabling this research. We acknowledge the funding through the VLAIO spin-off fellowship HBC.2018.2092 and through the EU within the H2020-EE-2016-RIA-IA programme for the project ‘Model Predictive Control and Innovative System Integration of GEOTABS;-) in Hybrid Low Grade Thermal Energy Systems - Hybrid MPC GEOTABS’ (grant number 723649). We acknowledge the university of Leuven (KU Leuven) for funding the development and valorization track of white-box MPC through the C3 project C32/18/063. This work emerged from IBPSA Project 1, an international project conducted under the umbrella of the International Building Performance Simulation Association. Project 1 develops and demonstrates a BIM/GIS and Modelica Framework for building and community energy system design and operation.

References

Åkesson, J. et al. (2010). “Modeling and optimization with Optimica and JModelica.org – Languages and tools for solving large-scale dynamic optimization problems”. In: *Computers & Chemical Engineering* 34.11, pp. 1737–1749. DOI: 10.1016/j.compchemeng.2009.11.011.

Andersson, Joel A. E. et al. (2019). “CasADi: a software framework for nonlinear optimization and optimal control”. In: *Mathematical Programming Computation* 11.1, pp. 1–36. DOI: 10.1007/s12532-018-0139-4.

Blum, D. et al. (2019). “Prototyping the BOPTEST framework for simulation-based testing of advanced control strategies in buildings.” In: *Proceedings of the 16th International Conference of IBPSA*. Rome, Italy.

Drgoňa, Ján et al. (2020). “All you need to know about model predictive control for buildings”. In: *Annual Reviews in Control* 50, pp. 190–232. ISSN: 1367-5788. DOI: 10.1016/j.arcontrol.2020.09.001.

Frank, Stephen et al. (2016). “Hybrid model-based and data-driven fault detection and diagnostics for commercial buildings”. In: *2016 ACEEE Summer Study on Energy Efficiency in Buildings*. Pacific Grove, CA.

International Energy Agency (2019). *Global Status Report for Buildings and Construction: Towards a Zero Emissions, Efficient and Resilient Buildings and Construction Sector*. Tech. rep.

Jorissen, F., W. Boydens, and L. Helsen (2017). “Validated air handling unit model using indirect evaporative cooling”. In: *Journal of Building Performance Simulation* 11.1, pp. 48–64. DOI: 10.1080/19401493.2016.1273391.

Jorissen, F., W. Boydens, and L. Helsen (2018). “TACO, an Automated Toolchain for Model Predictive Control of Building Systems: Implementation and Verification”. In: *Journal of Building Performance Simulation* 12.2, pp. 180–192. DOI: 10.1080/19401493.2018.1498537.

Jorissen, F., W. Boydens, and L. Helsen (2019). “Implementation and Verification of the Integrated Envelope, HVAC and Controller Model of the Solarwind Office Building in Modelica”. In: *Journal of Building Performance Simulation* 12.4, pp. 445–464. DOI: 10.1080/19401493.2018.1544277.

Jorissen, Filip (2018-04). “Toolchain for Optimal Control and Design of Energy Systems in Buildings”. PhD thesis. Arenberg Doctoral School, KU Leuven.

Jorissen, Filip, Glenn Reynders, et al. (2018). “Implementation and Verification of the IDEAS Building Energy Simulation Library”. In: *Journal of Building Performance Simulation* 11.6, pp. 669–688. DOI: 10.1080/19401493.2018.1428361.

Jorissen, Filip, Michael Wetter, and Lieve Helsen (2018). “Simplifications for Hydronic System Models in Modelica”. In: *Journal of Building Performance Simulation* 11.6, pp. 639–654. DOI: 10.1080/19401493.2017.1421263.

KU Leuven and 3E (2012). *IDEAS*. https://github.com/open-ideas/IDEAS.

Sturzenegger, David et al. (2016). “Model Predictive Control of a Swiss Office Building: Implementation, Results, and Cost-Benefit Analysis”. In: *IEEE Transaction on Control Systems Technology* 24.1, pp. 1–12. DOI: 10.1109/TCST.2015.2415411.

Wetter, Michael et al. (2014). “Modelica Buildings library”. In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506.

Wetter, M et al. (2019-09). “IBPSA Project 1: BIM/GIS and Modelica framework for building and community energy system design and operation – ongoing developments, lessons learned and challenges”. In: *IOP Conference Series: Earth and Environmental Science* 323, p. 012114. DOI: 10.1088/1755-1315/323/1/012114.

Software Architecture and Implementation of Modelica Buildings Library Coupling for Spawn of EnergyPlus

Michael Wetter[1] Kyle Benne[2] Baptiste Ravache[1]

[1]Lawrence Berkeley National Laboratory, Berkeley, CA
[2]National Renewable Energy Laboratory, Golden, CO

Abstract

Spawn of EnergyPlus is a next-generation energy simulation engine that targets control design and implementation workflows. Spawn reuses the weather, lighting, loads, and envelope modules from EnergyPlus through a precompiled library and couples them with HVAC and control models implemented in Modelica. Thus, for Spawn, the EnergyPlus HVAC models are removed. Spawn has been designed to perform coupled simulation with any number of EnergyPlus models, supporting simulation of a single building or multiple buildings as part of a district energy system.

This paper describes how the Modelica objects are implemented and synchronized to allow the modular specification at the Modelica-level that uses a Functional Mock-up Unit (FMU) that contains the EnergyPlus model. A key feature of our implementation is that multiple instances of Modelica models call C functions, which jointly build a data structure that defines parameters, inputs and outputs of the EnergyPlus model. This data structure is used during the initialization to generate an FMU that contains a fully configured EnergyPlus model. This FMU is then accessed by all Modelica models to exchange with EnergyPlus values for parameters, inputs and outputs during the simulation. This setup allows the Modelica models to be instantiated in a modular, object-oriented manner, as is typical for Modelica, yet they jointly construct and use an FMU that contains EnergyPlus.

Compared to an HVAC and envelope simulation that uses a native Modelica building model of comparable level of detail, the Modelica-EnergyPlus model translates about 35% faster and simulates about 50% faster.

Keywords: Modelica Buildings Library, Spawn of EnergyPlus, Modelica External Object, FMI

1 Introduction

Modelica has been shown to be well suited to support research, development and design of building and district energy systems, including their control logic (Wetter and Treeck 2017; Wetter, Treeck, et al. 2019). These applications typically require coupled simulations of the energy system and the building envelope. Coupled simulation of building envelopes and energy systems has proven challenging for various reasons. Building envelope models such as the ones in the Modelica Buildings (Wetter, Zuo, Thierry S. Nouidui, et al. 2014), BuildingSystems (Nytsch-Geusen et al. 2013) and IDEAS (Jorissen, Reynders, et al. 2018) libraries add a significant amount of code and a correspondingly large number of continuous-time state variables. These result in long translation times as Modelica tools do not yet satisfactory exploit repeated structures to keep translation time reasonably short. For simulation, the envelope model introduces a large number of continuous time states. These present a problem for the implicit ordinary differential equation solvers that are typically used on these stiff problems as these solvers scale superlinearly in the number of states. At the same time, the use of explicit solvers requires careful model tuning, which is not practical for most users (Jorissen, Wetter, and Helsen 2015). Finally, porting envelope models to Modelica would require considerable resources for porting algorithms including for shading calculations, 3D heat transfer, and coupled heat and moisture transfer through building fabrics, many of which may be better implemented in traditional imperative code. Tools for converting 3D data models for the building envelope would also need to be adapted to support input for Modelica. While future efforts by different building simulation developers may proceed along these lines, more advances are needed in Modelica translators, and multi-rate solvers for systems of stiff ordinary differential equation need to be accessible from Modelica tools in order to make use of such models practical for simulation of large buildings.

The US Department of Energy (DOE) has sponsored the development of EnergyPlus, a whole building energy simulation program (Crawley et al. 2001), since 1996. EnergyPlus is built on fundamental assumptions that makes it poorly suited to modeling building control sequences as they are implemented in physical controllers. DOE has also sponsored the development of the Modelica Buildings Library which is well suited to model HVAC and controls, but suffers from scalability to large building models for the above mentioned reasons. Spawn of EnergyPlus (or just Spawn) is the latest whole-building energy simulation program sponsored by DOE. Developed by the National Labs and industry, Spawn reuses the EnergyPlus envelope model and couples it to Modelica HVAC and control models from the Modelica Buildings Library (Wetter, Benne, et al. 2020), thereby combining the strengths of the two

DOI
10.3384/ecp21181325

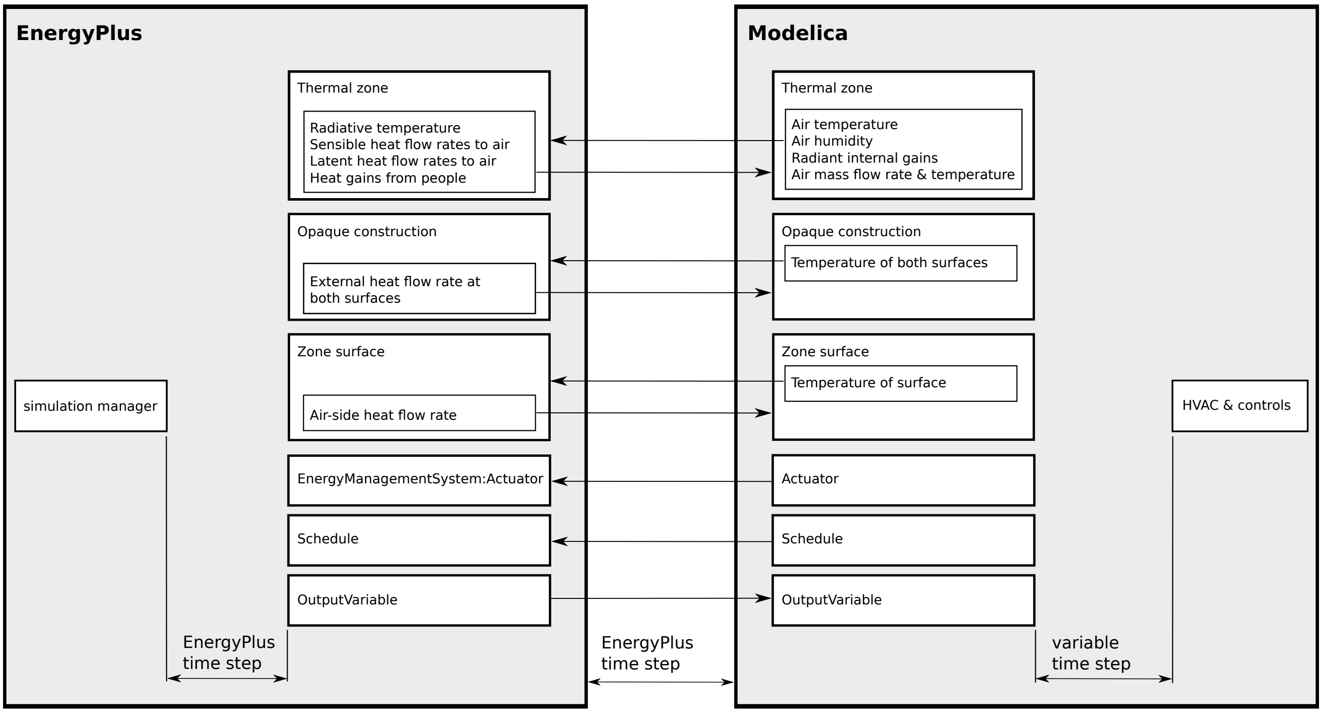


Figure 1. Partitioning of the envelope, room and HVAC model.

software approaches and implementations. Spawn is not an imminent replacement for EnergyPlus. Rather, it is intended to provide several capabilities that significantly advance beyond EnergyPlus and the Modelica Buildings Library. These include modeling of novel HVAC and district energy systems, scalable simulation of large buildings, simulation of control sequences represented in ways that also allow their implementation on building automation systems through a digitized control delivery process, and intrinsic support of multi-physics simulation and co-simulation with third party models.

This paper describes the additions to the Modelica Buildings Library that enables coupling Modelica models to the EnergyPlus envelope model in a way that automatically sets up the coupled simulation. While this implementation is specific for the coupling of building envelope models, a similar mechanism could be used to couple other models for building or district energy systems, e.g., aquifer thermal energy storage in which individual boreholes are connected to the same subsurface model.

The paper is structured as follows: Section 2 describes the variables that need to be exchanged between Modelica and EnergyPlus and states the requirements for coupled simulations. Section 3 describes the implementation. The key contribution is the mechanism that allows a deterministic synchronization of the execution of multiple instances of Modelica models. This synchronized execution is necessary for the software to collect all data required to generate one FMU for the whole building, before any Modelica model requests parameter values or output values from this FMU. Section 4 shows examples of the implementation, and Section 5 provides concluding remarks.

2 Requirements for Modelica Implementation

Figure 1 shows the variables that we require to be exchanged during the simulation between EnergyPlus and Modelica. The coupling variables connect Modelica thermal zone models, which implement the room air heat, mass and pressure balance, with the EnergyPlus thermal zone models that compute the convective heat gains from building fabrics and from internal loads. To support radiant systems, such as a radiant floors, coupling variables connect surface temperatures and heat flow rates between Modelica and EnergyPlus. Coupling variables are also used to read the values of EnergyPlus output variables for use in Modelica-implemented controllers, and to override EnergyPlus schedules and EnergyPlus Energy Management System actuators (Ellis, Torcellini, and Crawley 2007). The latter can be used to send signals to EnergyPlus to control non-HVAC elements such as an active facade, lighting, or other equipment that contributes to heat gains in the room and its surfaces.

To maximize usability and to enable drag-and-drop use in a graphical Modelica editor by non-experts in Modelica, the coupling needs to satisfy the following requirements.

1. To be able to graphically author and inspect models in a graphical modeling environment, each model (thermal zone, zone surface etc. as shown in Figure 1) should be its own instance, rather than being part of an array of models. This also ensures that translation and simulation diagnostics can be readily understood, which would not be the case if models

were referred to by an index of an array. Furthermore, if arrays of models were used, then wiring the connections would be impractical, in particular for large building models.

2. To enable simulation of multiple buildings, it should be possible to programmatically collect instances of models that belong to the same building in a hierarchical manner. It must also be possible to set common parameters centrally for all models that belong to the same building.
3. The coupled simulation between Modelica and EnergyPlus should be set up automatically for the user.
4. To allow large synchronization time steps, coupling should be done through slowly varying variables.

These requirements are addressed as follows. The first requirement is addressed by having individual Modelica classes (e.g., a **`model`** or **`block`**) for each object that communicates with EnergyPlus.

The second requirement is addressed by using **`inner`** /**`outer`** declarations of a building **`model`** that is used to set common parameters in the Modelica objects that communicate with the EnergyPlus building model. An example of common parameters is the name of the **`outer`** building declaration, which is used to determine which instances of thermal zone models belong to the same building.

The third requirement is addressed by adding a C layer to the Modelica Buildings Library, and a facility to the EnergyPlus program, that exports EnergyPlus as an FMU. This FMU is such that the required inputs and outputs, as specified by the Modelica instances, are exposed through its interface. This C layer invokes a command that generates the FMU, it loads the FMU, and it exchanges data with the FMU.

The second and third requirements leads to the situation that only after the Modelica model is partially initialized, the configuration of the FMU and hence the content of its `modelDescription.xml` file is known. Thus, the FMU needs to be generated during the Modelica initialization, and loaded before Modelica instances read parameter values from the FMU. This situation is a key reason for implementing our custom code for managing the FMU. This code is called using Modelica external C functions that are synchronized through the here explained mechanism.

The fourth requirement is addressed by modeling in Modelica the fast transients of the room air heat, mass and pressure balance, and coupling to EnergyPlus via the slower varying surface temperatures. This partitioning also has the advantage that the room air temperature, humidity and pressure, which are all connected to the HVAC system, are all natively implemented in Modelica. This allows using the same differential equation solver for these variables and the HVAC system.

We selected FMI for Model Exchange, version 2.0, rather than Co-Simulation because we allow certain signals to have direct feed-through. For example we allow setting a window blind and receiving the updated room daylight illuminance level at the same time step. This is only allowed in Model Exchange. Note, however, because EnergyPlus integrates its continuous time states using its own solvers, the FMU exposes no derivative. For Modelica, it looks like a discrete time model.

3 Implementation

3.1 Modelica Classes

For the coupling, we implemented the following Modelica classes:

`Building` Model that declares a building to which EnergyPlus objects belong to.

`ThermalZone` Model to connect to an EnergyPlus thermal zone.

`ZoneSurface` Model to exchange heat with an inside-facing surface of a thermal zone.

`OpaqueConstruction` Model to exchange heat with both surfaces of an opaque construction. The construction is modeled in Modelica. Heat is exchanged with the room-facing front surface and the back-side facing surface of an EnergyPlus construction.

`Actuator` Block to write to an EnergyPlus actuator.

`OutputVariable` Block to read an EnergyPlus output variable.

`Schedule` Block to write to an EnergyPlus schedule.

These Modelica classes allow communication between Modelica and EnergyPlus objects for thermal zones; thermal zone surfaces, either for the inside-facing surface only, or also its back-side facing surface (that may be located in an adjacent zone, or be the outside, or the ground temperature); Energy Management System (EMS) actuators; schedules; and output variables. To associate the Modelica classes to a building, the `Building` model is instantiated using the **`inner`** component prefix, and the other six classes use an instance of the `Building` model with the **`outer`** prefix. Through this mechanism, every instance that is in the instance tree below the `Building` instance will be associated with that particular building, and multiple buildings can be modeled in one Modelica model. All classes, other than `Building`, extend from `ExternalObject` to communicate with code implemented in C. In C, a data structure stores all building instances, and for each building instance, keeps track of which of the above objects belongs to that building instance. This data structure is set up when invoking the constructors of these Modelica instances via the `ExternalObject`. After all constructors are called, an FMU is generated for each building.

3.2 Constructor Synchronization

A key challenge was to enforce that all constructors are called *before* the FMU is generated. The Modelica Language Specification 3.5 does not guarantee that all constructors in a model are called before any Modelica function that uses a return value of a constructor is being used. In early research code, some Modelica tools invoked a function that uses the return value from the constructor of the `ExternalObject` before all instances called their constructor. As a consequence, the FMU exposed interface variables for some but not all instances, and the simulation terminated. Therefore, we changed the Modelica implementation to enforce the execution sequence shown in Algorithm 1.

Algorithm 1 Required execution sequence for generation and simulation of envelope model.

```
Data     Let $\mathscr{I}$ be the set of all instances that
         communicate with EnergyPlus.
Step 1:  For all instances $i \in \mathscr{I}$,
             call constructor for $i$.
Step 2:  For all instances $i \in \mathscr{I}$,
             initialize $i$.
             If first call to any initialization,
                 construct and load FMU,
                 setup experiment.
Step 3:  For all instances $i \in \mathscr{I}$,
             assign Modelica parameters by
             getting their values from the FMU.
Step 4:  For all instances $i \in \mathscr{I}$,
             at each synchronization step,
             set inputs, time and get outputs from FMU.
Step 5:  For all instances $i \in \mathscr{I}$,
             call destructor for $i$.
             If last call to any destructor
                 terminate and unload the FMU.
```

A key challenge was to enforce that in Algorithm 1, Step 1 is completed before Step 2 begins. While this would have been easy to enforce by using one constructor for the whole building model, such a centralized specification is impractical. To enforce this calling sequence, we therefore synchronized all objects using a `connector` that uses a potential and flow variable, together with `inner` and `outer` constructs that hide this complexity from the user. Note that these `inner` and `outer` constructs are different from the ones described in Section 3.1. Our implementation is based on the code provided by Beutlich (2021), which was motivated by Elmqvist et al. (2015).

Listing 1 to 9 describe this implementation, using a minimum representative example that has only one building and two thermal zones. The actual implementation is considerably larger and can be found in the Modelica Buildings Library 8.0.0, package `Buildings.ThermalZones.EnergyPlus`. Listing 1 shows the package with the `SynchronizeConnector` whose `flow` variable will be assigned by every thermal zone. The `SynchronizeConnector` is instantiated at the building level, as shown in Listing 2. The building sets its potential variable, which is needed for the model to be well defined, and it declares a variable `isSynchronized` whose value is set to the flow variable of the connector. Listing 3 shows the implementation of the thermal zone which extends `ObjectSynchronizer` and through this `extends` statement, gets a reference to the `outer building` and an instance of synchronization connector `synBui`. The call to `initialize` takes as an argument `building.isSynchronized`, which is computed by the `outer building` instance, and this computation requires the return value `nZ` of `initialize` which is assigned to `building.synchronize.done` via the `ObjectSynchronizer`. The other code in `ThermalZone` is a standard use of an external function interface that returns `adapter` which encapsulates a pointer to the C structure that contains the data structure needed to orchestrate the FMU coupling. This external function interface is shown in Listing 4. The two Modelica functions that communicate with the C implementation are shown in Listings 5 and 6, and the C implementation is shown in Listings 7 and 8.

Listing 1. Package that synchronizes all objects that belong to the building.

```
within BuildingRooms;
package Synchronize
  connector SynchronizeConnector
    Real do "Potential variable";
    flow Real done "Flow variable";
  end SynchronizeConnector;

  model SynchronizeBuilding
    SynchronizeConnector synchronize;
  end SynchronizeBuilding;

  model ObjectSynchronizer
    outer Building building;
    SynchronizeBuilding synBui;
  equation
  connect(building.synchronize,
      synBui.synchronize);
  end ObjectSynchronizer;
end Synchronize;
```

Listing 2. Model that declares building-level parameters.

```
within BuildingRooms;
model Building
  "Model that declares a building"
  Synchronize.SynchronizeConnector
      synchronize;
  Real synchronization_done =
      synchronize.done;
  Real isSynchronized;
equation
  synchronize.do = 0;
algorithm
  isSynchronized := synchronization_done;
end Building;
```

Listing 3. Model that implements the thermal zone.

```
within BuildingRooms;
model ThermalZone
```

```
  extends Synchronize.ObjectSynchronizer;
  constant String name=getInstanceName();
  ZoneClass adapter = ZoneClass(name,
      startTime);

  parameter Real startTime(fixed=false);
  parameter Integer nZ(
    fixed=false, start=0)
    "Total number of zones in building";
  constant Real k=1;
  Real tNext(start=startTime, fixed=true);
  Real T(start=293.15, fixed=true);
  Real Q_flow;

initial equation
  startTime=time;
  nZ=initialize(
    adapter=adapter,
    startTime=time,
    isSynchronized=building.isSynchronized)
      ;
equation
  when {initial(), time >= pre(tNext)} then
    (tNext, Q_flow) =exchange(
      adapter,
      time,
      T,
      nZ);
  end when;
  k*der(T) = Q_flow;
  nZ =synBui.synchronize.done;
end ThermalZone;
```

Listing 4. Model that implements the thermal zone.

```
within BuildingRooms;
class ZoneClass extends ExternalObject;

  function constructor
    input String name "Name of the zone";
    input Modelica.SIunits.Time startTime;
    output ZoneClass adapter;
  external "C" adapter=ZoneAllocate(name)
    annotation (
      Include="#include <thermalZone.c>",
      IncludeDirectory="modelica://
          BuildingRooms/Resources/C-Sources
          ");
  end constructor;

  function destructor
    input ZoneClass adapter;
  external "C" ZoneFree(adapter)
    annotation (
      Include="#include <thermalZone.c>",
      IncludeDirectory="modelica://
          BuildingRooms/Resources/C-Sources
          ");
  end destructor;
end ZoneClass;
```

Listing 5. Model that implements the thermal zone.

```
within BuildingRooms;
function initialize
  input ZoneClass adapter;
  input Real startTime;
  input Real isSynchronized;
  output Integer nZ "Number of zones";
  external "C" ZoneInitialize(adapter,
      startTime, nZ)
  annotation (
    Include="#include <thermalZone.c>",
    IncludeDirectory="modelica://
        BuildingRooms/Resources/C-Sources")
        ;
end initialize;
```

Listing 6. Model that implements the thermal zone.

```
within BuildingRooms;
function exchange
  input ZoneClass adapter;
  input Real t;
  input Real T;
  input Integer nZ;
  output Real tNext;
  output Real Q_flow;
  external "C" ZoneExchange(adapter, t, T,
      tNext, Q_flow)
  annotation (Include="#include <
      thermalZone.c>",
              IncludeDirectory="modelica://
                  BuildingRooms/Resources/
                  C-Sources");
end exchange;
```

Listing 7. Header file for C code that is a mock-up for the code that instantiates and communicates the FMU for the building envelope.

```
#ifndef thermalZone_h
#define thermalZone_h

typedef struct Zone{
  char* name;
} Zone;

#endif
```

Listing 8. C code that is a mock-up for the code that instantiates and communicates the FMU for the building envelope.

```
#ifndef thermalZone_c
#define thermalZone_c

#include <string.h>
#include <stdbool.h>

#include "thermalZone.h"

static int nZon = 0; /* Number of zones   */
static bool buildingIsInstantiated = false;

void* ZoneAllocate(const char* name){
  Zone* ptrZone;

   /* Allocate zone and assign name */
  ptrZone = (Zone*) malloc(sizeof(Zone));
  ptrZone->name =
    malloc((strlen(name)+1) * sizeof(char))
      ;
  strcpy(ptrZone->name, name);
  /* Increment counter for zones */
  nZon++;

  ModelicaFormatMessage(
    "Allocated zone %s\n", name);
  return (void*) ptrZone;
}

void ZoneInitialize(
    void* object,
    double startTime,
    int* nZ){
    Zone* zone = (Zone*) object;
    *nZ = nZon;
    if (!buildingIsInstantiated){
      /* Here, the actual implementation
         constructs an FMU that
         is shared by all zones.
         This requires that all zones
         executed ZoneAllocate().
      */
        buildingIsInstantiated = true;
        ModelicaFormatMessage(
          "Initialized zone %s.
              Instantiated building, nZ = %
              d.\n",
          zone->name, nZon);
    }
    else{
        ModelicaFormatMessage(
          "Initialized zone %s, nZon = %d\n
              ",
          zone->name, nZon);
    }
}

void ZoneExchange(
    void* object,
    double time,
    double T,
    double* tNext,
    double* Q_flow){
    Zone* zone = (Zone*) object;
    /* In the actual implementation,
       this is computed in an FMU.
    */
    *Q_flow = 283.15-T;
    *tNext = time + 1;
    ModelicaFormatMessage(
      "Exchanged with zone %s at time=%f,
          nZon = %d\n",
      zone->name, time, nZon);
}

void ZoneFree(void* object){
    Zone* zone = (Zone*) object;
    free(zone->name);
    free(zone);
}

#endif
```

For the user, the complexity of the synchronization is hidden. A building and its elements can be configured using the same Modelica constructs as are used for other instances, as Listing 9 shows.

Listing 9. Model that instantiates a building and two thermal zones that belong to this building.

```
within BuildingRooms;
model MyBuildingInstance
  "Building with two thermal zones, e.g.,
      nZ=2"
  inner Building building;
  ThermalZone t1;
  ThermalZone t2;
end MyBuildingInstance;
```

Simulating this model will give an output such as

```
Allocated zone MyBuildingInstance.t2
Allocated zone MyBuildingInstance.t1
Initialized zone MyBuildingInstance.t1.
    Instantiated building, nZ = 2.
Initialized zone MyBuildingInstance.t2,
    nZon = 2
Initialized zone MyBuildingInstance.t1,
    nZon = 2
Initialized zone MyBuildingInstance.t2,
    nZon = 2
Exchanged with zone MyBuildingInstance.t1
    at time=0.000000, nZon = 2
Exchanged with zone MyBuildingInstance.t2
    at time=0.000000, nZon = 2
...
```

3.3 C API

To control the FMU that contains the EnergyPlus envelope model, we developed a library in C which uses the FMI Library (*FMI Library* 2021) to interact with the FMU. Figure 2 shows the UML sequence diagram. Each Modelica object that communicates with EnergyPlus extends from the Modelica built-in class `ExternalObject`. Through its constructor, the Modelica instance calls the C code which registers the object in a `static struct`, and stores parameters that are declared in Modelica. These parameters include for example the name of a thermal zone so that it can be matched to the thermal zone object in the EnergyPlus model. Through the name of the **`outer`** instance of `Buildings`, objects that belong to the same building are registered accordingly. After all constructors are called, the first call to `initialize` will invoke a program that generates the FMU. Next, through the Modelica function `getParameters`, parameters such as the volumes of a thermal zone that are computed by EnergyPlus are retrieved from the FMU and assigned to Modelica parameters. During the simulation, the Modelica `exchange` function exchanges data and synchronizes time with the FMU. Finally, the destructor of the Modelica `ExternalObject` terminates and unloads the FMU.

3.4 FMU Generation

During the `initialize` step, an executable program `spawn` is invoked to generate a unique FMU for each `Building` configuration. `spawn` is invoked via a command line interface, which accepts a JSON file that specifies the contents of the resulting FMU. All configuration is specified by the Modelica classes described in Section 3.1.

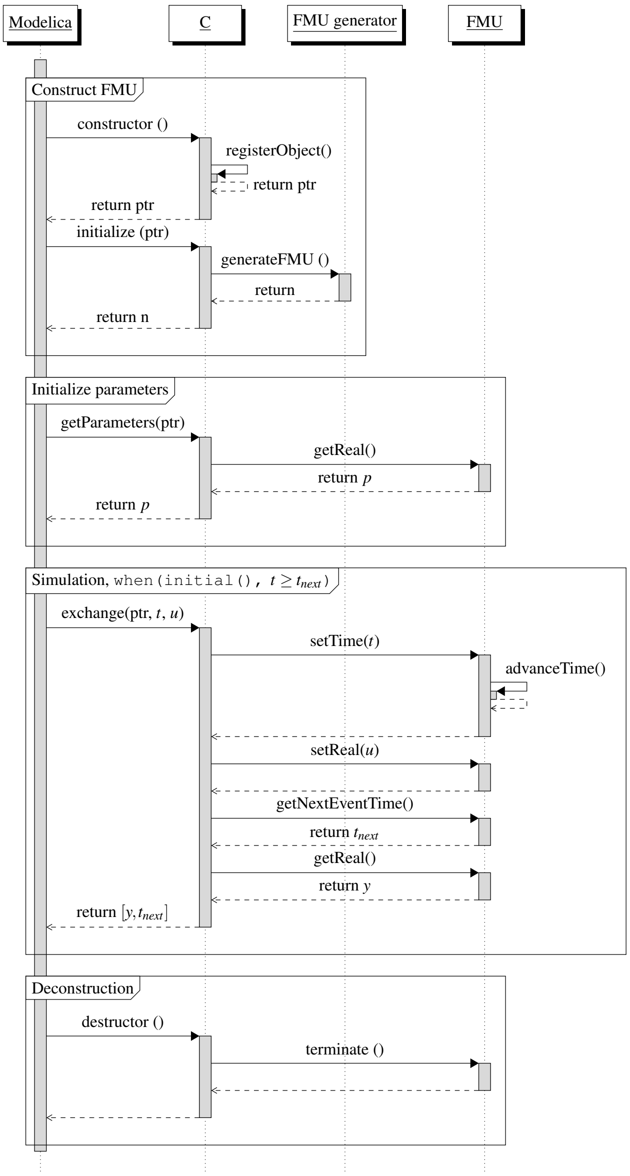


Figure 2. Sequence diagram for interaction with FMU.

The following steps are taken by the `spawn` program during FMU generation.

1. Create a staging directory.
2. Copy required resources into the staging directory, including the EnergyPlus input data file (IDF), and weather files. These files are specified by the user via parameters of the Modelica `Building` model.
3. Modify the given EnergyPlus IDF file so that it conforms to Spawn's requirements. The primary modification is to remove any EnergyPlus HVAC and control related objects.
4. Copy a custom EnergyPlus based shared library into the staging directory.
5. Generate a `modelDescription.xml` file, according to the variables that are requested via JSON input.
6. Compress the staging directory into zip format.

The resulting FMU is a self contained package with all of the resources required for an EnergyPlus based building simulation. Although it is possible to interact with the command line tool directly, it is currently not supported as a stand-alone tool.

3.5 Changes to EnergyPlus

The coupling of EnergyPlus with Modelica introduces unique requirements that EnergyPlus did not originally address. First, EnergyPlus was its own simulation manager and controlled the progression of simulated time; in Spawn, time is managed by Modelica. Second, although EnergyPlus included an External Interface feature for runtime data exchange, the existing capability was insufficient for Spawn. Most importantly, the External Interface feature limited the communication step to that of the zone time step, a limitation that derived from EnergyPlus' internal HVAC and control system models. In Spawn, HVAC and control system models are simulated using Modelica.

We modified EnergyPlus to allow the EnergyPlus HVAC loop to be bypassed, leaving the core EnergyPlus zone heat balance calculation engine which, based on our modifications, can be invoked at any simulated time even below the traditional zone time step limit of one minute.

We also added a software layer on top of EnergyPlus that facilitates simulation in which EnergyPlus is advanced through time by another program, and data is exchanged at each step. The former External Interface approach to co-simulation was based on a client-server architecture operating over a TCP/IP socket. However, socket based communication involves serialization and de-serialization at the endpoints that introduces a performance penalty with every exchange. The new approach implemented for Spawn is based on a co-routine design pattern. The co-routine is implemented using two threads. One thread contains the conventional EnergyPlus routine, and a second thread is a control thread that implements functions such as `setTime`, `setReal`, and `getReal`. In the co-routine, only one thread is active at any moment in time, and the two threads share memory, making data exchange between them efficient. The co-routine works by ping-pong'ing between the two threads. The EnergyPlus thread is blocked until a signal from the control thread is sent to advance in time; in turn the control thread is blocked until EnergyPlus signals that it has advanced to the desired time. When the EnergyPlus thread is blocked, the control thread can access EnergyPlus state and respond to requests for data. This results in an efficient data exchange with EnergyPlus and limited modifications to EnergyPlus code. This software layer combined with EnergyPlus is compiled as a shared library and included in the generated FMU described in Section 3.4.

4 Examples

We will now show two examples. The first example shows how translation and simulation time compares between a scalable model that uses an identical Modelica HVAC and control model with the envelope model of either the Modelica Buildings Library (Wetter, Zuo, and Thierry Stephane Nouidui 2011; Thierry Stephane Nouidui et al. 2012) or of EnergyPlus. The second example shows how to configure a Modelica model that uses the EnergyPlus envelope model to control a shade.

4.1 Translation and Simulation Time

This example shows how translation and simulation time changes between a native Modelica implementation and the EnergyPlus-Modelica coupled implementation for a building model with detailed HVAC system of varying size. For this example, we created a scalable model of the Modelica Buildings Library's `ThermalZones.Detailed.MixedAir` thermal zone model and the EnergyPlus envelope model `ThermalZones.EnergyPlus.ThermalZone`. Both cases model multiple floors that are representative of the large office building from the commercial reference building models for Chicago, IL (Deru et al. 2011). Each floor has 4 perimeter zones and a large core zone. Each floor is served by its own VAV system that includes an economizer, heating and cooling water-to-air coils and terminal reheat boxes. The system controls the ventilation, heating and cooling of all five zones based on ASHRAE Guideline 36 (ASHRAE 2018). Both cases use the same HVAC model. The hot- and cold-water loops are modeled with idealized heat sources and sinks.

The template models are scaled in size by varying the number of floors as shown in Table 1. As each floor is served by one HVAC system and has 5 thermal zones, the case with 10 floors has, for example, 10 HVAC systems and 50 thermal zones.

To have different state trajectories for each floor, each floor was configured to have a slightly different design air flow rate. This measure ensures that each floor triggers state events that are not simultaneous to state events from other floors, and that the adaptive time step solver computes indeed different error estimates for each floor, which overall may lead to more time steps as the number of diverse floors increases. Without this measure, the scaling may have been non-representative as temperatures in different floors typically evolve on different trajectories.

The models are available from `https://github.com/lbl-srg/modelica-buildings`, commit 15b90ae8bd5c4f3d6de23eee66b2efaab0c78b60.[1] The translated model with 10 thermal zones has 1700 continuous states and 48800 time varying variables if the `MixedAir` model is used, and 810 continuous states (about half of the native Modelica implementation) and 36800 time varying variables if the `EnergyPlus` model is used. All models were simulated for the days indicated in Table 1, using the Chicago TMY3 weather file. We used Dymola 2021 on Ubuntu 18.04 with the CVode solver, a tolerance of 10^{-5} and the sparse solver unless indicated otherwise in the table.

Table 1 show the translation and simulation times. The simulation time corresponds to the total CPU time required to simulate the compiled model. [2] Figure 3 shows the CPU time as a function of model time, and the relative computing time for each day. As can be seen in the figure, the slope of the CPU time is not constant over the model time. These change in slope are attributed to the change in dynamics of the state trajectories that occurs during certain parts of the model time. As shown in the plot, there are no step changes in the CPU time. A step change would have indicated a numerical problem, which may distort the total computing time as the numerical error is not an artifact of the different envelope model but rather of the resulting differential algebraic system of equations.

[1] Modelica package Examples.ScalableBenchmarks.ZoneScaling.

[2] This version of Spawn computes the numerically expensive shadow calculations from January 1 to the start day of the simulation. For the cases where the simulation starts in summer, this time is substantial. Because this is planned to be corrected in future releases, we subtracted this time in all reported results.

Table 1. Translation and simulation time for the `MixedAir` and EnergyPlus thermal zone model.

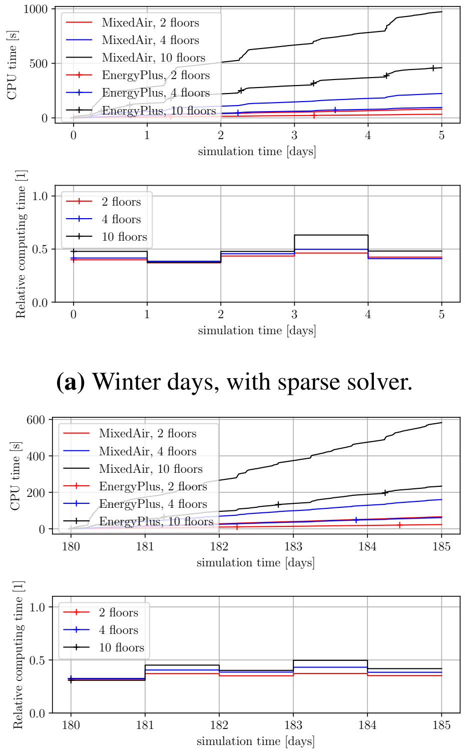
Model	Floors	Translation Time [s]	Simulation Time [s]
MixedAir days 1-5	2	47	81
	4	89	223
	10	230	973
EnergyPlus days 1-5	2	31 (66%)	35 (43%)
	4	57 (64%)	97 (43%)
	10	139 (61%)	462 (48%)
MixedAir days 1-5, non-sparse	2		102
	4		361
	10		2570
EnergyPlus days 1-5, non-sparse	2		39 (38%)
	4		115 (32%)
	10		677 (26%)
MixedAir days 180-185	2		66
	4		160
	10		583
EnergyPlus days 180-185	2		30 (45%)
	4		74 (46%)
	10		264 (45%)

In summary, the models with the EnergyPlus thermal zones translate about 35% faster. Their simulation time is also about 50% faster for the cases with the sparse solver. For the model with 10 zones, disabling the sparse solver increases the computing time by a factor of 2.5 for the case with the `MixedAir` model, and by about 1.5 for the case with the `EnergyPlus` model.

4.2 Shade Control

This example illustrates how to interface with EnergyPlus from different Modelica models. Figure 4 shows a model of a building with three thermal zones, one of which has a window with a shade. These are simulated in EnergyPlus. Modelica models the window shade control sequence, for which it obtains incident solar radiation from EnergyPlus and sends the actuation signal back to EnergyPlus. Modelica also models the fresh air supply and an idealized cooling system in each thermal zone. The air heat and mass balances for each room are modeled in Modelica, and the envelope heat transfer is modeled in EnergyPlus.

In the figure, the instance `building` specifies building-level settings, such as the EnergyPlus IDF file. The three blue icons in the middle connect to three EnergyPlus thermal zones. The instance `incBeaSou` reads from EnergyPlus the incident beam solar radiation on the window, and the instance `actSha` actuates the window shade. In the EnergyPlus model, the west-facing thermal zone has a window blind that is open if its control signal is 0 or closed if it is 6. The control sequence obtains the room air temperature of the west-facing zone from the Modelica instance `zonWes`, and connects it to a hysteresis block that switches its output to `true` if the zone temperature is above 24°C, and to false if it drops below 23°C. The instance `incBeaSou` obtains from EnergyPlus the incident solar beam radiation on the outside of the window, and feeds it into a hysteresis block that outputs `true` if its in-



(a) Winter days, with sparse solver.

(b) Summer days, with sparse solver.

Figure 3. CPU time and relative computing time for Modelica Buildings Library MixedAir and EnergyPlus thermal zone model. The relative computing time is the ratio of CPU time it took to simulate the indicated day for Spawn compared to the native Modelica model.

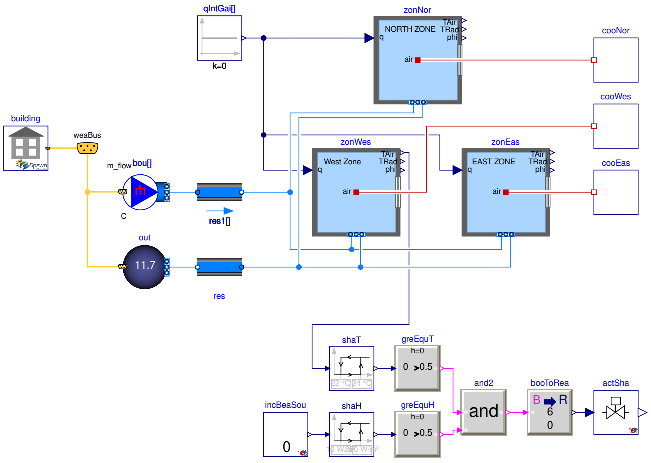


Figure 4. Schematic diagram of the Spawn model with shade control, available from the Buildings Library as `Buildings.ThermalZones.EnergyPlus.Examples.SingleFamilyHouse.ShadeControl`. Note that the thermal zone models `zon*`, the output variable reader for the incident solar radiation `incBeaSou` and the actuator for the shade `actSha` all communicate with the same EnergyPlus model via C functions. Thus, the control loop from shade control to zone temperature `zonWes.TAir` is closed via EnergyPlus.

put exceeds 200 W/m^2, and switches to `false` if it drops below 10 W/m^2. The instance `actSha` connects to the actuator in EnergyPlus that activates this shade. If both outputs of the hysteresis blocks are `true`, then the EnergyPlus shade actuator is deployed by setting the input of `actSha` to 6. Otherwise, the input is set to 0. To the right of the model, there are three idealized cooling systems that keep the room air temperature below 25°C in each of the three zones. Also, each zone is connected to a constant, unconditioned outside air supply.

5 Conclusions

Through the use of **`inner`**/**`outer`** constructs and a **`flow`** variable, we were able to ensure a correct synchronization of Modelica models that communicate with a common data structure via C functions that each use a distinct pointer to memory obtained through a Modelica `ExternalObject`. This was essential for enabling model authoring in the same way as one typically does with Modelica models that are instantiated in a distributed manner within a larger Modelica system model. The implementation ensures that each constructor is called before the first Modelica instance calls its initialization function that generates and imports the FMU, which is then accessed for input and output by the different Modelica instances. The resulting implementation allows simulating one or several buildings, where each building is represented by an FMU that can have any number of objects that are synchronized with Modelica.

For a Modelica model that consists of a variable air volume flow system and detailed control sequence, coupled to a multi-zone building envelope model that is implemented either in Modelica or in EnergyPlus, the version that uses EnergyPlus translates about 35% faster and simulates about 50% faster.

Acknowledgements

This research was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy, under Contract No. DE-AC02-05CH11231.

The authors would also like to thank Thomas Beutlich for pointing out and adapting a similar synchronization problem from Elmqvist et al. (2015).

References

ASHRAE (2018-06). *ASHRAE Guideline 36-2018 – High Performance Sequences of Operation for HVAC systems.*

Beutlich, Thomas (2021). *Modelica Specification issue 2842.* URL: https://github.com/modelica/ModelicaSpecification/issues/2842#issuecomment-776194950 (visited on 2021-03-08).

Crawley, Drury B. et al. (2001). “EnergyPlus: creating a new-generation building energy simulation program”. In: *Energy and Buildings* 33.4. Special Issue: BUILDING SIMULATION’99, pp. 319–331. ISSN: 0378-7788. DOI: https://doi.org/10.1016/S0378-7788(00)00114-6.

Deru, Michael et al. (2011-02). *U.S. Department of Energy Commercial Reference Building Models of the National Building Stock*. Tech. rep. National Renewable Energy Laboratory.

Ellis, Peter G., Paul A. Torcellini, and Drury B. Crawley (2007). “Simulation of Energy Management Systems in EnergyPlus”. In: *Proc. of the 10-th IBPSA Conference*. Ed. by Jiang Yi et al. International Building Performance Simulation Association and Tsinghua University. URL: http://www.ibpsa.org/.

Elmqvist, Hilding et al. (2015-09). “Generic Modelica Framework for MultiBody Contacts and Discrete Element Method”. In: *11-th International Modelica Conference*. Ed. by Peter Fritzson and Hilding Elmqvist. Modelica Association. Paris, France, pp. 427–440. DOI: 10.3384/ecp15118427.

Jorissen, Filip, Glenn Reynders, et al. (2018). “Implementation and Verification of the IDEAS Building Energy Simulation Library”. In: *Journal of Building Performance Simulation* 11 (6), pp. 669–688. DOI: 10.1080/19401493.2018.1428361.

Jorissen, Filip, Michael Wetter, and Lieve Helsen (2015-09). “Simulation Speed Analysis and Improvements of Modelica Models for Building Energy Simulation”. In: *11-th International Modelica Conference*. Ed. by Peter Fritzson and Hilding Elmqvist. Modelica Association. Paris, France, pp. 59–69. DOI: 10.3384/ecp1511859.

FMI Library (2021). URL: https://github.com/modelon-community/fmi-library (visited on 2021-03-08).

Nouidui, Thierry Stephane et al. (2012-09). “Validation and Application of the Room Model of the Modelica Buildings Library”. In: *Proc. of the 9-th International Modelica Conference*. Modelica Association. Munich, Germany, pp. 727–736. DOI: 10.3384/ecp12076727.

Nytsch-Geusen, Christoph et al. (2013). “Modelica BuildingSystems eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme”. In: *Bauphysik* 35.1, pp. 21–29. ISSN: 1437-0980. DOI: 10.1002/bapi.201310045.

Wetter, Michael, Kyle Benne, et al. (2020-09). “Lifting the Garage Door on Spawn, an Open-Source BEM-Controls Engine”. In: *Proc. of Building Performance Modeling Conference and SimBuild*. Chicago, IL, USA, pp. 518–525. URL: https://simulationresearch.lbl.gov/wetter/download/2020-simBuild-spawn.pdf.

Wetter, Michael and Christoph van Treeck (2017-09). *IEA EBC Annex 60: New Generation Computing Tools for Building and Community Energy Systems*. ISBN: 978-0-692-89748-5. URL: http://www.iea-annex60.org/pubs.html.

Wetter, Michael, Christoph van Treeck, et al. (2019-09). “IBPSA Project 1: BIM/GIS and Modelica framework for building and community energy system design and operation – ongoing developments, lessons learned and challenges”. In: *IOP Conference Series: Earth and Environmental Science* 323, p. 012114. DOI: 10.1088/1755-1315/323/1/012114.

Wetter, Michael, Wangda Zuo, Thierry S. Nouidui, et al. (2014). “Modelica Buildings library”. In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506.

Wetter, Michael, Wangda Zuo, and Thierry Stephane Nouidui (2011-11). “Modeling of heat transfer in rooms in the Modelica "Buildings" library”. In: *Proc. of the 12-th IBPSA Conference*. International Building Performance Simulation Association. Sydney, Australia, pp. 1096–1103. URL: https://simulationresearch.lbl.gov/wetter/download/2011-ibpsa-BuildingsLib.pdf.

Coupling physical and machine learning models: case study of a single-family house

Basak Falay[1] Sandra Wilfling[2] Qamar Alfalouji[2] Johannes Exenberger[2] Thomas Schranz[2]
Christian Møldrup Legaard[3] Ingo Leusbrock[1] Gerald Schweiger[2]

[1]AEE-Institue for Sustainable Technologies, Austria b.falay@aee.at
[2]Institute of Software Technology, Technical University of Graz, Austria gerald.schweiger@tugraz.at
[3]DIGIT, Department of Electrical and Computer Engineering, Aarhus University, Denmark, cml@ece.au.dk

Abstract

The emergence of Cyber-Physical Systems poses new challenges for traditional modelling and simulation techniques. We need to combine white, grey, and black box models as well as different tools developed for specific subsystems and domains. Co-simulation is a promising approach to modeling and simulating such systems. This paper presents a case study where a physical model of a building's heating system implemented in Modelica is co-simulated with a machine learning model of a stratified hot water tank implemented in Python. The Python model is exported as Functional Mock-up Unit using UniFMU.

Keywords: Co-Simulation, Functional Mock-Up Interface, Modelling, Machine Learning

1 Introduction

Future intelligent and integrated energy systems must have a high degree of flexibility and efficiency to ensure reliable and sustainable operation (Lund et al. 2017). Along with the rapid expansion of renewable energy, this degree of flexibility and efficiency can be achieved by overcoming the clear separation between different sectors and by increasing connectivity and the associated data availability through the integration of sensors and edge/fog computing (Vatanparvar and Faruque 2018). All of these developments drive the transition towards so-called Cyber-Physical Energy Systems (Palensky, Widl, and Elsheikh 2013). Cyber technologies (sensors, edge/fog computing, IoT networks, etc.) can monitor the physical systems, enable communication between different subsystems, and control them. Thus, the emergence of Cyber-Physical Systems poses new challenges for traditional modelling and simulation approaches.

One of these challenges is that models need to combine computational systems and data communication networks with physical systems. Furthermore, recent studies showed that pure white-box models based on first principles deal with drawbacks such as time-consuming development, validation problems or low computational speed (Li and Wen 2014). Consequently, these approaches have limited use for complex systems such as intelligent buildings outside of academia (Schweiger, Nilsson, et al. 2020). Black-box approaches examine the system from the outside using input/output relations. Depending on the approach, they are computationally efficient but compared to white-box approaches they lack in generalizability and extensibility (Thieblemont et al. 2017). Beside white-box and black-box models, grey-box models fall in between (Harish and Kumar 2016). Several papers highlighted the importance of combining white-, grey-, and black-box models for analyzing and optimizing Cyber-Physical Systems (O'Dwyer et al. 2019; Killian and Kozek 2016; Thilker, Madsen, and Jørgensen 2021).

There are two options to simulate the interactions between subsystems; (i) the entire system can be modelled and simulated with a single tool referred to as monolithic, (ii) already established models for the respective subsystems are coupled in co-simulation (Gomes et al. 2018). A recent survey discussed the advantages, disadvantages, and challenges of co-simulation approaches (Schweiger, Engel, et al. 2018). This survey showed that experts consider the Functional Mock-Up Interface (FMI) standard to be the most promising standard for continuous-time, discrete-event, and hybrid co-simulation.

In this paper, the physical parameters of a subsystem (stratified storage tank) are not available. In this situation, model calibration and parameter estimation approaches can be used depending on availability of the measurement data. On the other hand, machine learning models can be as well used to mimic the behavior of the system by construct relationships between input and outputs without being dependent on the components parameters. Artificial Neural Networks was used to model the stratified storage tank in (Géczy-Vig and Farkas 2010). In this work, Random Forest (RF) was used to model the temperatures in each layer of the stratified storage tank. Since the states of the other components influence the state of the stratified storage, we have created a co-simulation workflow where the machine learning and physical models can be coupled. Physical and machine learning models are available at https://github.com/tug-cps/NextHyb2 . Unfortunately, we cannot publish the data due to data privacy policy. Therefore, we have additionally generated a synthetic, open-source data set.

DOI
10.3384/ecp21181335

2 Method

2.1 Heating System of Single-family House

A single-family house with $180m^2$ floor area, located in Austria with an annual energy consumption of 7500 kWh was analyzed in this work. Figure 1 gives an overview of the main components of the single-family house heating system. The house, equipped with a floor heating system, has three different heat sources: (i) a solar collector with $46m^2$ flat plate area, (ii) a stove which directly heats the house, and the excess heat feeds the storage tank and (iii) an air-to-water heat pump. Additionally, an estimated $3m^3$ storage tank bridges these three heat sources in order to increase the efficiency of the heating system. The house has an indoor pool ($24m^3$), which is heated by the hot water storage tank or directly by the solar collector.

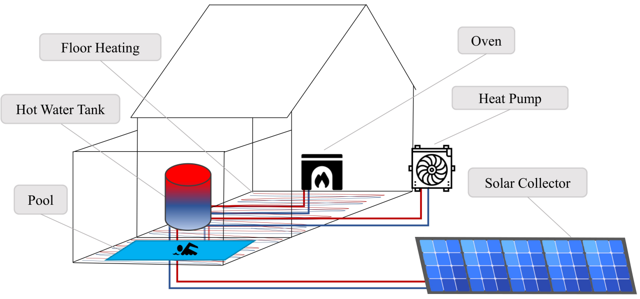


Figure 1. Overview of the single-family house heating system. Red line represents the supply and blue line represents the return temperatures.

The following rule-based control strategy of the heating system is given below.

- The priority of the solar collector is to maintain the temperature of top layer of the storage tank at 52°. If this condition is satisfied, then the excess heat from the solar collectors heats the indoor pool to 35°.
- If the solar collector cannot meet the heating demand of the indoor pool and if the top layer temperature of storage tank is higher than 52°, the storage tank heats the pool.
- If these conditions don't satisfy or the temperature of the bottom layer of the storage tank drops below 35°, the heat pump turns on.
- If the temperature of the stove is higher than 40°, the excess heat is fed into the storage tank.

2.2 Measurement Data

In Appendix, Figure A.1 shows an overview of the heating system components and the locations of the heat meters. Temperatures are represented in ($T_{component}$, mass flow rate in $dm_{component}$. Four temperature sensors from top to bottom respectively $T_{Storage,1}, T_{Storage,2}, T_{Storage,3}, T_{Storage,4}$ are located at the storage tank. The measured data from the heat meters is between 01.02.2019 and 31.01.2020, with a temporal resolution of 1 minute. Figure 2 gives an overview of the data quality of the measurement data. White lines represent the missing data points and corresponding periods. 4% of the measurement data is missing; 65% of them falls into the period between November 2019 and January 2020, 25% of them falls into September 2019. In addition to missing values, there are wrong measurements between November 2019 and January 2020 due to the failures in the meters.

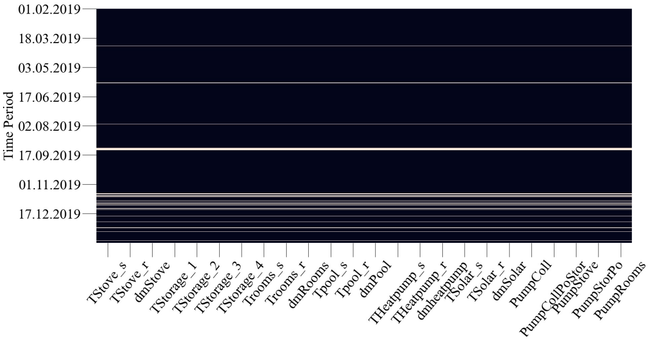


Figure 2. Missing data periods for the given measurement data

The missing parts of the data were imputed by taking the profile of the previous day. Figure 3 demonstrates the imputation of missing data points for four days in a row, given in dashed lines. The measurement data was ignored after November 2019 due to the bad quality of data. The whole data set was resampled to 15-minute values to avoid the over fitting the predictions of the ML model. After post-processing, the dataset had 27840 datapoints. The resampled data was later used for training and testing for the ML model.

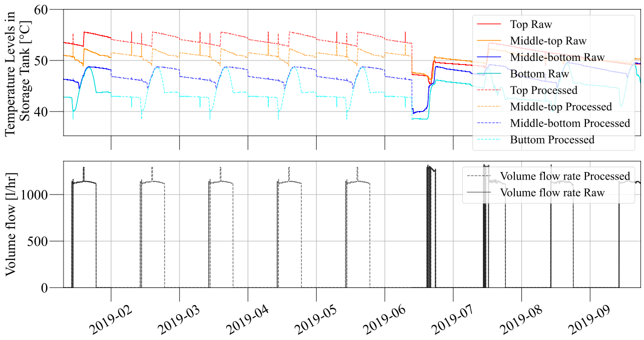


Figure 3. Imputation of the missing data

One of the most critical features in the dataset is mass flow rates from each component. Figure 4 shows the sparsity of the mass flow rates from each components. The y-axis represent the total data points (27840) after pre-processing, the x-axis represents the mass flow values of the components. The black points in the figure show the values that are not zero and the gaps between the black points represent the zero values. Mass flow rate in the stove has the highest percentage 99.8% of zero values and

DOI
10.3384/ecp21181335

the mass flow rate in the *CollectortoPool* has the lowest percentage with 86%.

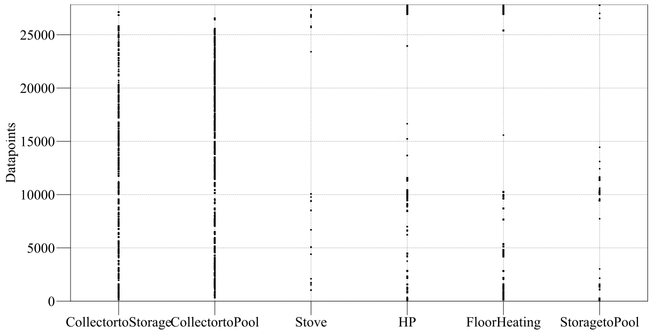


Figure 4. Visualization of data sparsity in mass flow rates of each component

2.3 Physical Models

The physical models were implemented in the Modelica language (Fritzson and Engelson 1998). Modelica is an open source, a-causal, object-oriented and multi-domain modelling language. A discussion of limitations and promising approaches of the Modelica language can be found (Schweiger, Nilsson, et al. 2020). All the models used in this system are based on the Modelica IBPSA Project 1 library (Wetter, Treeck, et al. 2019) and the Buildings Library (Wetter, Zuo, et al. 2014). Dymola was used to simulate Modelica models (Brück et al. 2002). The following sub - implemented in Modelica are: Solar system (`Buildings.Fluid.SolarCollectors.EN12975`), heat pump (`Buildings.Fluid.HeatPumps.CarnotTCon`) and the indoor pool (`Modelica.Fluid.Vessels.ClosedVolume`). The energy demand of the house and the heat supply profile of the stove were taken from the measurement data instead of modelling these components. Since there was no weather profile acquired within the given data period, Typical Meteorological Year 3 (TMY3) for Austria were generated from Meteonorm.

2.4 Machine Learning Model

There was no available information of the system parameters of the storage tank such as the insulation material and the thickness, the wall thickness, the height or the locations of the temperature sensors. Therefore, the storage tank was modelled based on RF. RF is a combination of tree predictors which splits nodes based on a best split of random subsets of the features, thus reducing the variance of the tree model and increasing the overall predictive power of the model.

The RF model predicted the four temperature layers of the storage tank. An overview of the input features for the model is given in Figure 5. The static input features are temperatures, T_i, and mass flow rates, dm_i, from the solar collector, heat pump, floor heating and stove. The dynamic features are the four temperature layers of the storage tank with a 1-hour look-back time with interval 15 minutes and 15 minutes prediction horizon, see Figure 6. The measured data was split randomly into training (80%, 50 epochs) and testing (20%). The model hyper-parameters are n_estimators = 100 which represents the number of decision trees that achieves the best trade-off between the accuracy and efficiency; max_depth that has been set to an unlimited value so the nodes can expand automatically; and min_samples_split = 2. The implementation was done using the Python framework presented in (Schranz et al. n.d.) based on Scikit-learn.

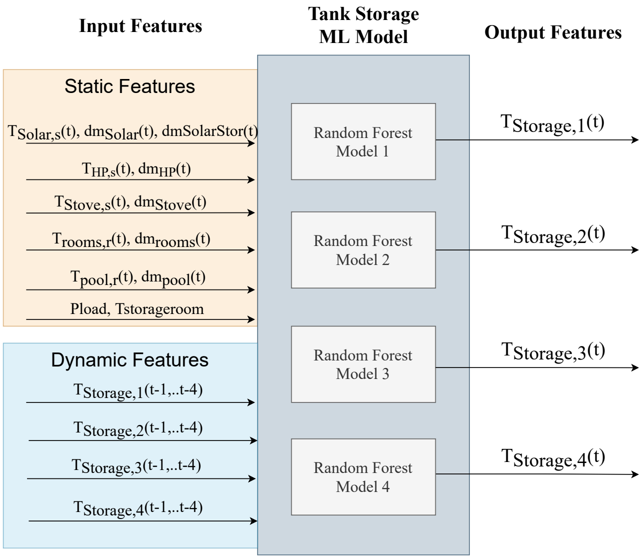


Figure 5. Input/Output features of the applied machine learning model of the storage tank.

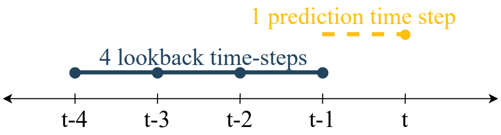


Figure 6. At time t, four look-back time-steps are used to predict one time-step in future with each step = 15 minutes.

2.4.1 Model Performance Analysis

Two criteria were selected to evaluate the performance of the RF model: the coefficient of variation of the Root mean square error (CVRMSE) and mean absolute percentage error (MAPE) given in Equation 1 and Equation 2.

$$CV(RMSE) = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2}}{Y} * 100 \qquad (1)$$

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{|Y_i - \hat{Y}_i|}{Y_i}\right) * 100 \qquad (2)$$

where Y is the true value, $\hat{Y}$ is predicted value, $\bar{Y}$ is the average of the true values over N test samples.

2.5 Co-Simulation

To integrate the ML model of the storage tank into the simulation environment, the ML model must be exported as an FMU. The UniFMU tool (Legaard et al. 2021) was used to generate a Python-based FMU. Therefore a template of the FMU was generated using the command `unifmu generate python name`.

To specify the behavior the dummy example in Listing 1, implemented by the generated FMU, is replaced with the components of the ML model. A benefit of this is that the `scikit-learn` code can be reused and integrated into the FMU gradually. This makes sure that no breaking changes occur. A crucial part of the FMUs implementation is the `fmi2DoStep` method which instructs the model to simulate forward in time for an amount of time corresponding to the time step. For the storage-tank FMU, this is equivalent to running one or more inference steps of the trained model.

Listing 1. Implementation of `fmi2DoStep` by storage model.

```
from sklearn.ensemble import
    RandomForestRegressor
from sklearn.datasets import
    make_regression
...
def do_step(current_time,step_size,
    no_step_prior):
     self.temp_next=self.forrest(self.
         temp_prevs)
return Fmi2Status.ok
```

3 Results and Discussion

3.1 Validation of the ML model

Table 1 shows the model performance on the test data set. The ML model is imported as FMU in Dymola. Testing of the FMU-ML model with the measurement data is performed in Dymola environment, see Figure 7. $T_{Storage,4}$ and $T_{Storage,3}$ are the worst predicted target value according to the CVRMSE and MAPE. The discussion of Table 1 is supported with the results of the testing.

Table 1. Performance metrics of predicting the four target temperature values: $T_{Storage,1}$, $T_{Storage,2}$, $T_{Storage,3}$ and $T_{Storage,4}$ using random forest models

	CVRMSE	*MAPE*
$T_{Storage,1}$	0.0097	2.3056
$T_{Storage,2}$	0.011	2.2656
$T_{Storage,3}$	0.0157	4.9064
$T_{Storage,4}$	0.0281	6.0215

Winter, spring and summer periods were chosen, aiming to represent different boundary conditions. The only difference in each test period was the initial values set for the FMU-ML. These initial values of the static and dynamic input features were chosen from the measurement data based on each period starting time. Since the stratified hot water storage tank is a short term storage, the daily predictions are representative. In these three figures, 9 days period for each season was chosen to show the model prediction based on interactions of all heating supplies. In Figure 8, Figure 9 and Figure 10, the first subplot represents the comparison between the true and the predicted temperature values of each 4 layers of the storage tank. The true temperature values of the top layer, middle-top layer, middle bottom layer and the bottom layer are respectively red, dark orange, blue and cyan dashed lines. The predicted temperature values are represented the same color code but in straight lines. In the second subplot, the mass flow rates from different components are given. The mass flow rates stand for when the specific component is turned on/off.

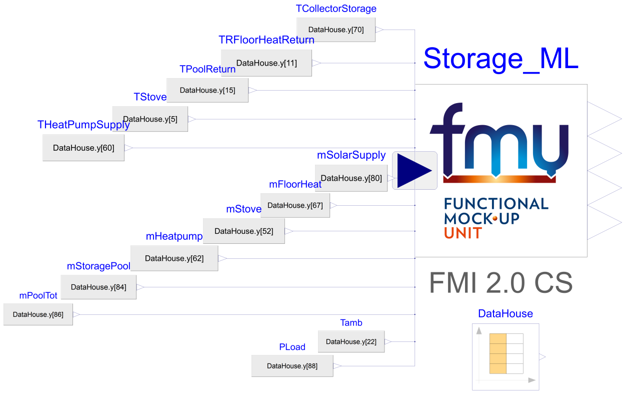


Figure 7. Dymola layout of testing FMU-ML storage model

Figure 8 represents the frequently running components; solar collector and pool heating for summer period. Based on these components inputs, the ML model shows good aggrement with the measurement data during the summer period. One of the static input features, "PLoad", which indicates whether at least one component in the system is on, is introduced to capture the cooling behavior of the storage tank. It is observed in the summer period during the night when there is no load, the four temperature values of the storage tank decrease.

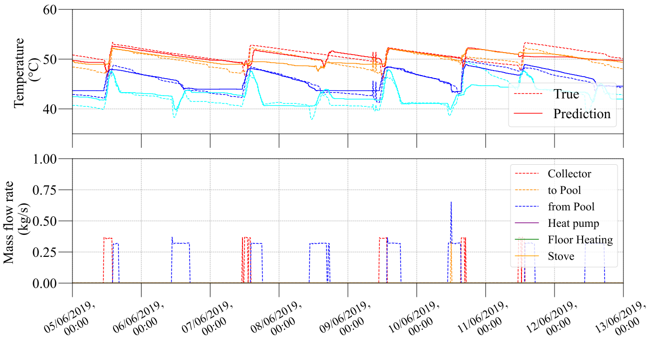


Figure 8. Storage temperature levels predicted vs measurement in summer period

The cold return water from the floor heating is fed into the storage tank from the bottom and the middle layer. These temperatures are expected to decrease as in the measurement data. In Figure 9, between 18th and 19th March, when the floor heating starts, the predictions of the temperature of the bottom layer fails. On the 19th March, when the heat pump is on, represented in the purple line, the middle bottom temperature shows an increasing behavior. However, it cannot reach to the values of the measurement data. On 20th March, only two components are on as in the summer period. On this day, the prediction of the temperature values can catch the measurement data.

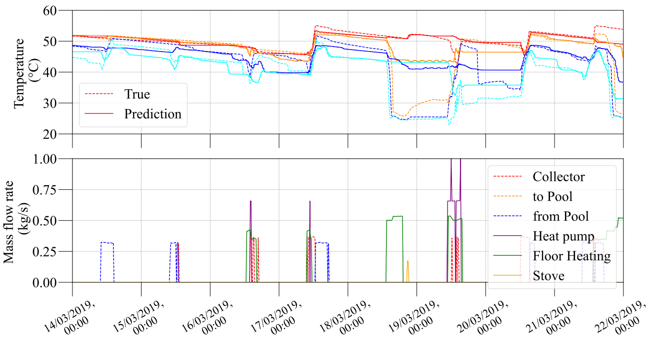


Figure 9. Storage temperature levels predicted vs measurement in spring period

The winter period is selected between February and March due to lack of winter representation from the data, explain in Section 2.2. The bottom temperature layer of the storage tank shows a decreasing behavior due to the floor heating as in the measurement data. However, due to the control strategies when the heat pump turns on, the middle bottom temperature of the storage tank doesn't increase as in the measurement data profile.

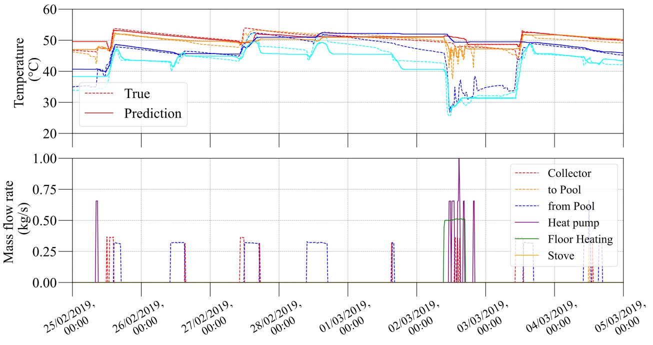


Figure 10. Storage temperature levels predicted vs measurement in winter period

From these three testing periods of the FMU-ML model, it is observed that the temperature values of the storage tank is predicted better when there is only collector component is on. In Figure 4, data that represent collector to storage and collector to pool is denser than the other components data. Therefore, these static input features can dominate the predictions more than the other static features which are sparse. Additionally, the dynamic features of the past predicted temperature values of the storage tank are as well input features. Once these values are predicted wrong, the error accumulates to the further time steps. Results from these tests also show the the performance of the $T_{Storage,4}$ and $T_{Storage,3}$ are worse than the other predicted target values.

3.2 System simulation

In Appendix, Figure A.2 shows the system implementation in Dymola. All heating components explained in usecase are framed with dashed lines in the figure. The figure is visually simplified by hiding the source/sink component inside of the 'StorageML_FMU' component where the supply or return fluid from each component is fed to storage tank. All the simulations are run in a virtual Ubuntu environment with 188 GB RAM and the Intel Xeon Silver 4215R CPU @ 3.2GHz CPUs. Dymola 2021 FD01 with Dassl solver and 10e-6 tolerance was used during this study. The system simulation with the FMU-ML was run 10 times. The averages of the CPU times for the 32 days simulation with 15 minutes interval is 228 seconds. The CPU-time taken to calculate one grid interval highly depends on how the ML algorithm is implemented, number of inputs features, number of processors.

The translated model statistics are given in Table 2. The originally described system has 1966 non-trivial DAEs, after translation it is reduced to an ODE system with 42 continuous time states.

Table 2. Model statistics:Translated model statistics of the single-family house with the FMU-ML storage component.

	FMU-ML Model
Constants	1733
Parameters depending	654
Continuous time states	42
Time varying variables	777
Alias variables	1342
Sizes of linear system of equations	{5}
Sizes after manipulation of the linear system of equations	{0}
Sizes of nonlinear system of equations	{6, 5, 3, 1, 1}
Sizes after manipulation of the nonlinear system of equations	{1, 1, 1, 1, 1}
Number of numerical Jacobians	0

Figure 11 shows the results of coupling ML and physical models of the single-family house heating system. The first subplot in Figure 11 shows the temperature levels of the tank for a 32 days period. The second subplot shows which component is switched on and the third shows the load condition for the storage tank. Despite the same real-world control strategies implemented into system, weather profile that represents the the measurement data is not available. The TMY3 from Meteonorm is used

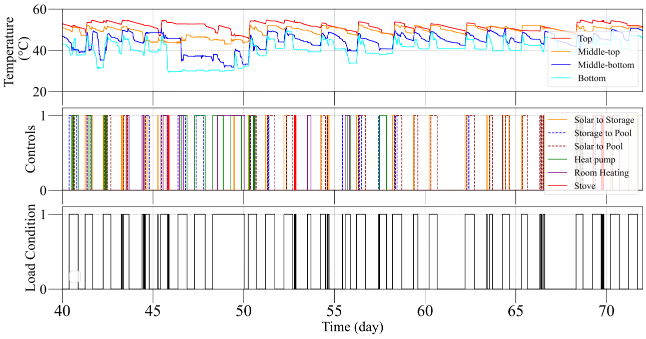


Figure 11. Storage temperature levels according to the controls

for the system simulation and solar collector provides different outputs than the measurement data. And all outputs based on the control strategies changes. Also the physical model parameters of the storage tank have not been adjusted to give a good comparison. Therefore, the results from FMU-ML system simulation cannot be compared with the measurement data.

Acknowledgements

The presented research is part of the project “NextHyb2” (FFG project number 881150) financed by the Austrian Research Promotion Agency FFG. We are grateful to Klima Energie Fonds for sharing the data. We also would like to acknowledge the contributions and the fruitful discussions of Carles Ribas Tugores and Walter Becke in the context of the use case study.

References

Brück, Dag et al. (2002). “Dymola for multi-engineering modeling and simulation”. In: *Proceedings of modelica*. Vol. 2002. Citeseer.

Fritzson, Peter and Vadim Engelson (1998). “Modelica—A unified object-oriented language for system modeling and simulation”. In: *European Conference on Object-Oriented Programming*. Springer, pp. 67–90.

Géczy-Vig, P. and I. Farkas (2010). “Neural network modelling of thermal stratification in a solar DHW storage”. In: *Solar Energy* 84, pp. 801–806. ISSN: 2261-236X. DOI: 10.1051/matecconf/201822501015.

Gomes, Cláudio et al. (2018). “Co-simulation: a survey”. In: 51.3. ISSN: 0360-0300. DOI: 10.1145/3179993.

Harish, VSKV and Arun Kumar (2016). “A review on modeling and simulation of building energy systems”. In: *Renewable and sustainable energy reviews* 56, pp. 1272–1292.

Killian, Michaela and Martin Kozek (2016). “Ten questions concerning model predictive control for energy efficient buildings”. In: *Building and Environment* 105, pp. 403–412.

Legaard, Christian Møldrup et al. (2021). “A Universal Mechanism for Implementing Functional Mock-up Units”. In: *11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. SIMULTECH 2021. Virtual Event, to appear.

Li, Xiwang and Jin Wen (2014). “Review of building energy modeling for control and operation”. In: *Renewable and Sustainable Energy Reviews* 37, pp. 517–537.

Lund, Henrik et al. (2017). “Smart energy and smart energy systems”. In: *Energy* 137.2, pp. 556–565. DOI: 10.1016/j.energy.2017.05.123.

O’Dwyer, Edward et al. (2019). “Smart energy systems for sustainable smart cities: Current developments, trends and future directions”. In: *Applied energy* 237, pp. 581–597.

Palensky, Peter, Edmund Widl, and Atiyah Elsheikh (2013). “Simulating cyber-physical energy systems: Challenges, tools and methods”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 44.3, pp. 318–326. DOI: 10.1109/TSMCC.2013.2265739.

Schranz, Thomas et al. (n.d.). “Energy Prediction under Changed Demand Conditions:Robust Machine Learning Models and Input Feature Combinations”. In: *Building Simulation 2021. International Building Performance Simulation Association*.

Schweiger, Gerald, Georg Engel, et al. (2018). “Co-simulation an empirical survey: applications, recent developments and future challenges”. In: *MATHMOD 2018 Extended Abstract Volume*, pp. 125–126.

Schweiger, Gerald, Henrik Nilsson, et al. (2020). “Modeling and simulation of large-scale systems: A systematic comparison of modeling paradigms”. In: *Applied Mathematics and Computation* 365, p. 124713.

Thieblemont, Hélène et al. (2017). “Predictive control strategies based on weather forecast in buildings with energy storage system: A review of the state-of-the art”. In: *Energy and Buildings* 153, pp. 485–500.

Thilker, Christian Ankerstjerne, Henrik Madsen, and John Bagterp Jørgensen (2021). “Advanced forecasting and disturbance modelling for model predictive control of smart energy systems”. In: *Applied Energy* 292, p. 116889.

Vatanparvar, Korosh and Mohammad Abdullah Al Faruque (2018). “Control-as-a-Service in Cyber-Physical Energy Systems over Fog Computing”. In: *Fog Computing in the Internet of Things: Intelligence at the Edge*. Ed. by Amir M. Rahmani et al. Cham: Springer International Publishing, pp. 123–144. ISBN: 978-3-319-57639-8. DOI: 10.1007/978-3-319-57639-8_7. URL: https://doi.org/10.1007/978-3-319-57639-8_7.

Wetter, Michael, C van Treeck, et al. (2019-09). “IBPSA Project 1: BIM/GIS and Modelica framework for building and community energy system design and operation – ongoing developments, lessons learned and challenges”. In: vol. 323. IOP Publishing, p. 012114. DOI: 10.1088/1755-1315/323/1/012114. URL: https://doi.org/10.1088/1755-1315/323/1/012114.

Wetter, Michael, Wangda Zuo, et al. (2014). “Modelica Buildings library”. In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506. URL: https://doi.org/10.1080/19401493.2013.765506.

A Appendix

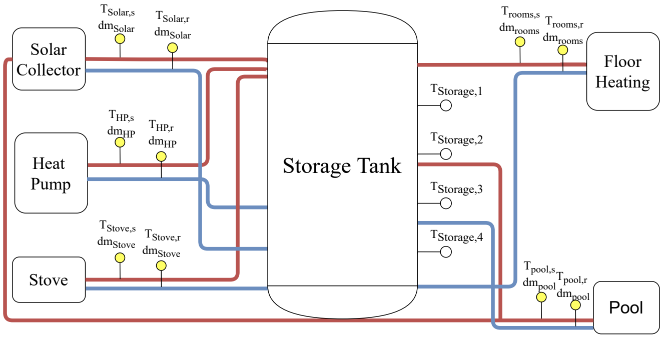


Figure A.1. Overview of the system hydraulic flow. The supply pipe is represented in red, return pipe in blue. In each pipe, the temperature and mass flow rates are measured.

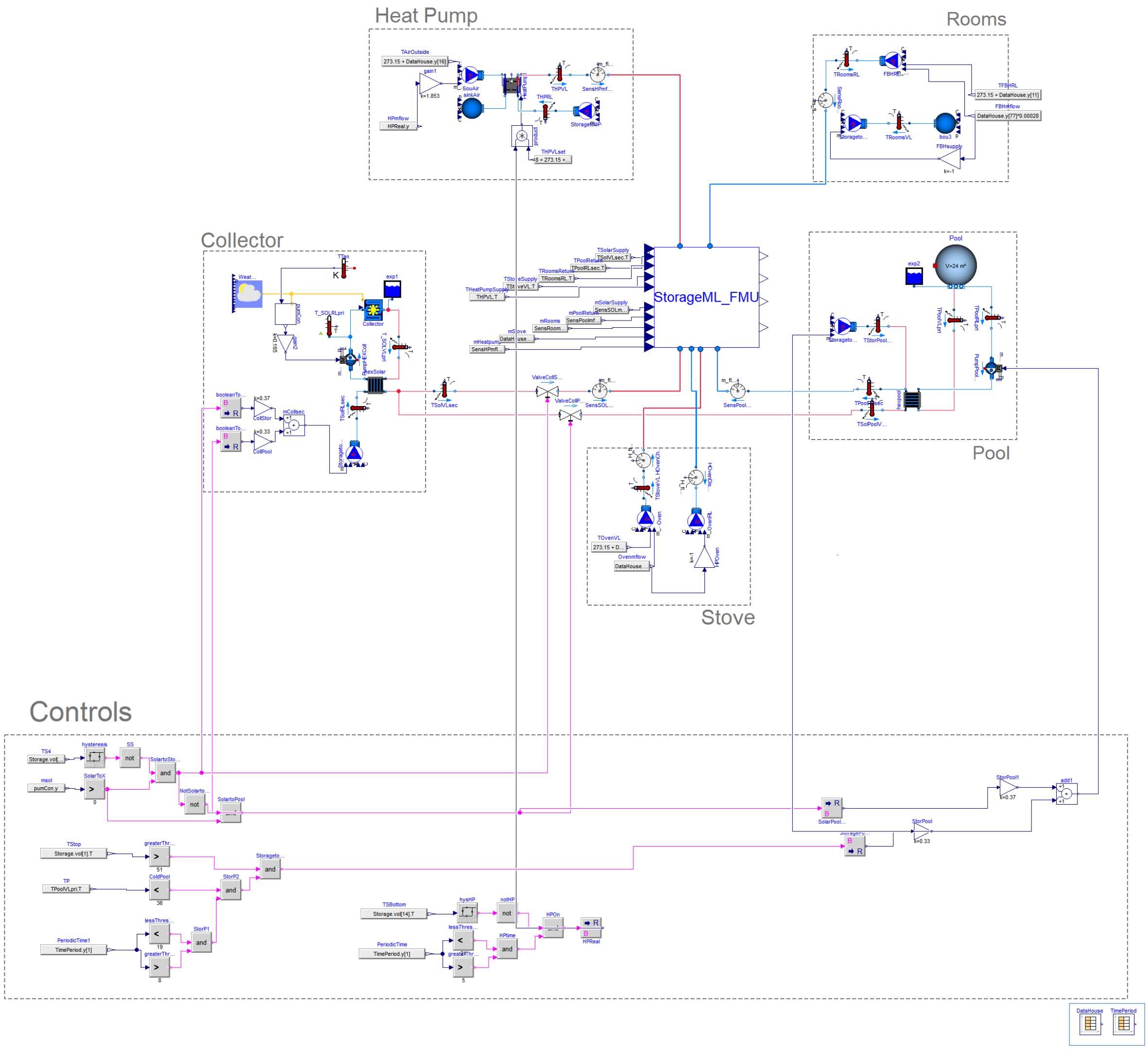


Figure A.2. Dymola layout of the single-family house heating system

Underfloor heating system model for building performance simulations

Stephan Göbel[1] Elaine Schmitt[1] Philipp Mehrfeld[1] Dirk Müller[1]

[1]Institute for Energy Efficient Buildings and Indoor Climate, RWTH Aachen University, Germany,
stephan.goebel@eonerc.rwth-aachen.de

Abstract

The efficiency of heat pump systems is highly dependent on the temperature gap between the sink and the source side. Therefore, it is necessary to accurately model the sink side to enable the most accurate and holistic analysis of building energy systems. In both residential and non-residential buildings, underfloor heating systems are becoming more and more widely used. The use of underfloor heating lowers the flow temperature of the heating system compared to a radiator, which increases the efficiency of a heat pump system. This paper provides an underfloor heating system model including automatic parametrization according to the European standard for surface embedded heating and cooling systems EN 1264. Since the model represents a whole underfloor system, it consists at the system level of the distributor and several heating circuits and takes the heat transfer at the smallest level from a pipe element through different floor layers into account. The model is verified for the system requirements according to European standard EN 1264. A parameter study with a variety of different underfloor heating parameters and floor layers shows that reductions in heat transfer in the underfloor heating system are compensated by an increase in the flow temperature. The highest influence on the temperature level of the system is exerted by the pipe spacing T, which raises the flow temperature by up to 10.9 K, from 36.6 °C ($T = 100$ mm) to 47.5 °C ($T = 400$ mm). The model is freely available on GitHub:
https://github.com/RWTH-EBC/AixLib/tree/issue890_HOMProject_FloorHeating

Keywords: Building performance simulation, EN 1264, automatic parametrization

1 Introduction

The accurate modelling of underfloor systems is important since the combination with heat pumps is on the rise and their efficiency is very sensitive to the temperature gap between the source and the sink side. While the source temperature is largely determined by the environment, the sink temperature is significantly influenced by the heat transfer system. Thus, the heat transfer system influences the efficiency of the heat pump and the efficiency of the entire building energy system. In a holistic analysis of a building energy system, accurate modelling of the heat sink is therefore always of great importance. Underfloor heating systems as an already widespread type of heat transfer systems offer several advantages over conventional radiator systems.

A great advantage is a uniform temperature distribution in the room due to a large transfer surface. Thus, underfloor heating systems transfer about two-thirds of the heat flow to the room by radiation and one-third by convection (*Taschenbuch für Heizung + Klimatechnik 13/14* 2012). Due to the high amount of thermal radiation, there is an increase in the temperature of the surrounding room surfaces, which leads to higher comfort in the room. From an energy perspective, underfloor heating systems require lower flow temperatures compared to conventional radiators which increases the potential of heat pumps. Furthermore, the possibility of passive cooling in combination with geothermal energy or even active cooling are promising methods in times of increasing cooling demand.

Hot water underfloor heating sytems according to EN 1264-1 (2021) are widely used. In this type of heating systems, water-flowing pipes (or other hollow sections) are laid in the floor (Hestermann et al. 2010). The European standard EN 1264 provides guidelines for surface embedded heating and cooling systems for residential and non-residential buildings and focusses on systems for thermal comfort. The standard also specifies standardized product characteristics by testing and calculation the thermal output of heating for technical specifications and certification. Additionally, the standard only describes systems that are attached directly or by means of fasteners to the structural base of the perimeter surfaces of the building. The EN 1264 series is devided in five parts:

- Part 1: Definitions and symbols
- Part 2: Floor heating: Methods for the determination of the thermal output using calculations and experimental tests
- Part 3: Dimensioning
- Part 4: Installation
- Part 5: Determination of the thermal output

To the best of the authors' knowledge, there are no models for underfloor heating in the known Modelica libraries

DOI
10.3384/ecp21181343

that allow a quick and easy parameterization according to EN 1264 and fit the modelling approach of the AixLib library. Borrajo Bastero et al. (2019) indicates a model and validation of a specific building. The focus of his work is the modeling of a specific building energy system. The design of the underfloor heating is not the main focus. Thus, the length of each heating circuit of the system and the associated pressure drop are not explicitly calculated. The underfloor heating system model of Weitzmann et al. (2005) uses the finite volume method where the heat flow is considered only. Also in this work, the system is modeled for a specific building. The key advantage of this model is the validation against measured data. The model considers the floor construction in detail. A quick transferability to other buildings does not seem to be possible. Within this paper, a simulation model for an underfloor heating system is presented. The model represents a wet system and contains all components of an underfloor heating system. It consists of several heating circuits for the heating of the individual rooms and a heating circuit distributor. In addition, the floor layers below and above the heating pipes are implemented. The highlight of this work is the automatic parameterization according to EN 1264-3 (2020), which ensures a fast application of the developed model. To connect the model to a building, only the building parameters, such as the number of rooms, room size, specific heat load of each room, and wall construction have to be transferred to the model. Standard values such as the pipe diameter or the pipe spacing can optionally be overwritten. Afterward, the system is dimensioned using the design equations of the implemented standard.

In the following, we will describe the underfloor heating system model and its submodels for the discretized pipe element, the heating circuit, the room level and the heating circuit distributor in detail. Afterwards, we discuss the results and demonstrate the validity of the model. We also show the influence of different parameters, such as the pipe spacing or the floor layers on the return temperature. At the end of this paper, we summarize the work and give an outlook for further improvements.

2 Underfloor heating system model

The model represents a wet system and contains all components of an underfloor heating system. It consists of the distributor and several heating circuits to warm up individual rooms. For that, the floor layers that are located above and below the heating pipes which transfer the heat to the rooms, are implemented in detail. The design according to EN 1264-3 (2020) is part of the model and can thus calculate its design parameters.

The floor heating model has the title *UnderfloorHeatingSystem* and follows a hierarchical structure. This allows the parameters to be passed to the respective submodels and makes parameterisation necessary only at the top level. The submodels of the overall system can also be used individually and are not only executable in the overall model.

The focus is set on the comprehensibility of the model. In each level, therefore, only the equations are implemented, which are crucial for this level. As can be seen in Figure 1, the smallest element of the undefloor heating system model represents a discretized pipe element. This pipe element is connected several times in series on the next higher level to form a heating circuit. In order to realize several heating circuits within a room, a model with parallel heating circuits exists on the next level.

The overall system at the top level connects the individual heating circuits from the rooms in a heating circuit distributor to distribute the total mass flow to the individual circuits. The system level is also used to connect the building model and the energy system with the underfloor heating. Furthermore, it provides the design parameters for mass flow and supply temperature. The individual model setups are explained in more detail in the following chapters.

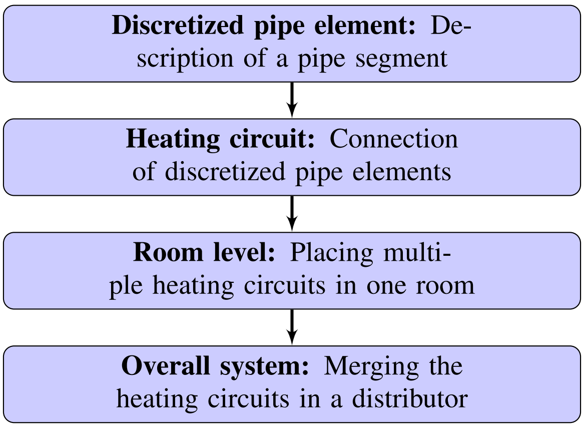

Discretized pipe element: Description of a pipe segment
Heating circuit: Connection of discretized pipe elements
Room level: Placing multiple heating circuits in one room
Overall system: Merging the heating circuits in a distributor

Figure 1. Hierarchical model structure of the developed underfloor heating system from the smallest pipe element to the complete system

2.1 Discretized Pipe Element

The discretized pipe element as the basic model of the underfloor heating system describes the thermal heat flow. Figure 2 offers an overview about the pipe element model. The aim of this model, called *UnderfloorHeatingElement*, is to divide a heating circuit into short pipe sections so that the decreasing fluid temperature within the pipe can be calculated in arbitrarily small steps.

Each pipe element is interpreted by a volume model with uniform fluid temperature, which represents the water volume in that discrete pipe section. The discretized pipe element model can calculate the water volume in two different ways. On the one hand it can use the inner diameter of the pipe, and the other it can use a time constant τ. In the model for the pipe element, the maximum value for the fluid velocity in the heating pipe is set to $0.5\,\mathrm{m\,s^{-1}}$. The user can decide whether exceeding this value will cause a warning or an error. If the water volume is calculated by means of time constant and the fluid velocity in the pipe

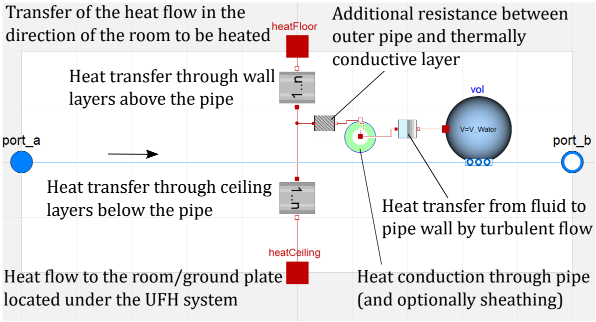

Transfer of the heat flow in the direction of the room to be heated
heatFloor
Additional resistance between outer pipe and thermally conductive layer
Heat transfer through wall layers above the pipe
1...n
vol
V=V_Water
port_a
port_b
Heat transfer through ceiling layers below the pipe
1...n
Heat transfer from fluid to pipe wall by turbulent flow
heatCeiling
Heat flow to the room/ground plate located under the UFH system
Heat conduction through pipe (and optionally sheathing)

Figure 2. Model view of the discretized pipe element in Dymola: The volume element "vol" contains the volume of water that is in heat exchange with the pipe walls and floor/ceiling layers; fluid and thermal quantities are calculated at the ports for connection to other quantities are calculated for the connection with other models

exceeds the maximum value of 0.5 m s^{-1}, the model gives the user a default value for the inner diameter for which the limit value is observed. If the parameter determination for the inner diameter is not intuitive for the user, this gives the possibility to determine a reasonable value.

The heat transfer through the floor layers is represented in the model *UnderfloorHeatingElement*, which means that it is also discretized in the overall model. This is necessary in order to generate the correct surface temperatures of the floor and ceiling and, consequently, the actual heat transfer into the rooms. Above a pipe section, there is thus exactly the floor surface that is heated by this pipe element.

Heat transfer through the discretized pipe element is divided into heat transfer through the pipe and heat conduction through the floor layers involved. From the inner pipe wall, heat is transferred by convection from the fluid onto the pipe wall and heat conduction through the pipe wall. For the interpretation of the heat conduction, the simplified model of the *AixLib CylindricHeatConduction* (Müller et al. 2016) was chosen, which calculates the steady-state heat conduction in the hollow cylinder. This keeps the parameterization simple for the user. Since the inertia of the system is primarily caused by the floor layers and the thermal conductivity for different pipe and sheathing materials is given in EN 1264-2 (2021) Annex D, this simplified way was chosen.

An additional resistance is implemented between the outer wall of the pipe and the floor layers. This can be interpreted primarily as the heat transfer between the pipe outer wall and the heat-conducting layer. This thermal resistance is indispensable because it is used to calculate the average temperature of the heat-conducting layer, which is transferred to the floor layers.

The heat conduction from the heat-conducting layer through the floor layers is divided into the layers above and below the floor heating. Therefore, there are two components which use the model *SimpleNLayer* from the *AixLib*. Those describe the transient heat conduction through these layers. The model *SimpleNLayer* is based on the consideration of two resistances and one capacitance per layer. The parameterization of the floor and ceiling layers is done by *Records*, analog to the High Order Model of *AixLib* (Constantin, Streblow, and Müller 2014). In these records, the material properties of the floor layers are stored, namely the density, thermal conductivity, heat capacity and the thickness. Since the floor and ceiling layers are already implemented in the underfloor heating model, it is necessary that they are bypassed or not included in the building model it is used within.

2.2 Heating Circuit

The model for a heating circuit in the floor heating system establishes the connection of several pipe elements. Different properties of the heating circuits in the system essentially determine the heat transfer to the heated room. Important parameters for determining the heating circuit structure are summarized in this model, called *UnderfloorHeatingCircuit*. The most important component of the heating circuit model is the arrangement of pipe elements connected in series, which are described by the model *UnderfloorHeatingElement* from chapter 2.1. Apart from that, the pressure losses and the average floor surface temperature in the room are determined at this level. Figure 3 shows the model's structure.

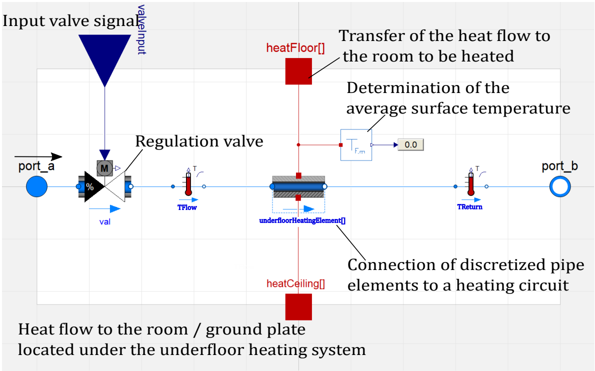

Input valve signal
valveInput
heatFloor[]
Transfer of the heat flow to the room to be heated
Determination of the average surface temperature
Regulation valve
0.0
port_a
M
val
TFlow
underfloorHeatingElement[]
TReturn
port_b
Connection of discretized pipe elements to a heating circuit
heatCeiling[]
Heat flow to the room / ground plate located under the underfloor heating system

Figure 3. Model view of a heating circuit in Dymola: The discretized pipe elements are connected to form a heating circuit and can be controlled via the regulating valve

The number of pipe elements within a heating circuit is specified by the user through the parameter *dis*. Temperature sensors are placed directly in front of the first and behind the last element to determine the flow and return temperature in each heating circuit. Furthermore, each heating circuit in the underfloor heating system is equipped with a control device. For that, a valve is placed in front of the first pipe element. The valve is designed as an equal-percentage two-way valve, which can regulate volume flows.

The nominal pressure loss is divided into the pressure loss caused by the pipe resistance and the valve. The pressure loss of the heating pipe *dp_Pipe* has a reference value of 100 $\mathrm{Pa\,m^{-1}}$, but can also be adjusted by the user. The pressure loss through the control device *dp_Valve* is based on the data for a heating circuit distributor from Schütz Energy Systems (2017). The model verifies that the limit value of 250 mbar according to EN 1264-3 (2020) is not exceeded.

To maintain comfort in rooms, the floor surface temperature T_F should not go beyond certain limits. The limits for the average floor surface temperature $T_\mathrm{F,m}$ are given in EN 1264-2 (2021). However, the user can also specify his limit value as a parameter. That the maximum value for the average surface temperature is not exceeded is also checked by the heating circuit model. With the mean surface temperature, the user keeps control over an important limit value in underfloor heating systems.

2.3 Room Level

The model *UnderfloorHeatingRoom* bundles the results of one heated room in the underfloor heating system. The main tasks of the model are the parallel connection of several heating circuit models that belong to one room and the calculation of the nominal mass flows of these heating circuits according to EN 1264-3 (2020). An overview of the model is shown in the Figure 4.

The maximum pipe length of heating circuits is between 80 m and 120 m depending on the manufacturer. To meet this requirement, the user can specify a maximum pipe length for the room level model. If the maximum specified pipe length is exceeded, another heating circuit is provided in the room whereas all existing circuits within a room are identically parameterized and designed. Accordingly, several identical heating circuits can be controlled individually via the vectorized valve presetting.

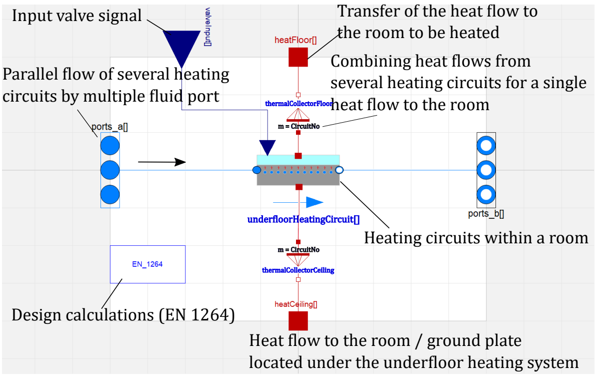

Input valve signal
valveInput[]
Transfer of the heat flow to the room to be heated
heatFloor[]
Parallel flow of several heating circuits by multiple fluid port
Combining heat flows from several heating circuits for a single heat flow to the room
thermalCollectorFloor
m = CircuitNo
ports_a[]
ports_b[]
underfloorHeatingCircuit[]
Heating circuits within a room
EN_1264
m = CircuitNo
thermalCollectorCeiling
heatCeiling[]
Design calculations (EN 1264)
Heat flow to the room / ground plate located under the underfloor heating system

Figure 4. Model view of the room level in Dymola for the underfloor heating system: Several heating circuits located in parallel in one room; According to EN 1264, the required system-dependent coefficients for the design are calculated

The system-dependent coefficients are calculated according to EN 1264-3 (2020). To determine these values, a submodel *EN_1264* is inserted into the room level model. In this submodel, the system-dependent coefficient K_H and the limit of specific thermal output q_G are calculated for further use at the room level. To determine these values, the necessary tables from EN 1264-2 (2021) are deposited.

The limit of specific thermal output q_G is used in particular to check the plausibility of the input parameters. If the value for q_G exceeds the maximum limit of specific thermal output $q_\mathrm{G,max}$, the model will report this with an error and the user needs to adjust his input parameters. Besides, the limit of specific thermal output is used to find out whether the underfloor heating is sufficient to cover the present heat load. If the calculated limit of specific thermal output is below the specific heat load q_des, further heat flow needs to be generated by other heating devices. The model *UnderfloorHeatingRoom* informs about this as well.

The *Records* for the floor and ceiling layers must be constructed correctly for the model to define screed and flooring as the layers above the heating pipe. The lower floor layers must consist of insulation, load-bearing substrate and plaster. If the ground plate is below the underfloor heating, there must be four layers below the underfloor heating. The top layer must then still be the insulation layer. The assignment of the values in the *Records* to the model is illustrated in Table 1. A correct definition of the layers in the data set is a prerequisite for a correct design of the underfloor heating system in the model.

In addition, compliance with the thermal resistance specifications for insulation according to EN 1264-4 (2021) Table 1 is checked at the room level. For rooms located above other heated rooms, the minimum for the insulation's thermal resistance is set at $R_{\lambda,\mathrm{Ins,min}} = 0.75\,\mathrm{m^2\,K\,W^{-1}}$, for adjoining unheated areas at $R_{\lambda,\mathrm{Ins,min}} = 1.25\,\mathrm{m^2\,K\,W^{-1}}$. In order to ensure a proper comparison between the actual and minimal values, the assignment of the floor layers from Table 1 must be observed.

2.4 Distributor

The top level of the model for an underfloor heating system includes the consolidation of all heating circuits in the distributor and is called *UnderfloorHeatingSystem*. The room level, which was represented in chapter 2.3, is inserted as a sub-model in the respective number of rooms to be heated. The overall model can be used to connect to the building and the energy system for detailed thermal building simulation.

A heating circuit distributor divides the total mass flow to the individual heating circuits and recombines them after passing through the room level in return. The flow and return temperatures are recorded by sensors. With the valve settings passed through from the individual heating circuit to the overall system, each heating circuit in the underfloor heating system can be controlled separately. The heat flows of the heated rooms are transferred from the underfloor heating to the rooms by means of discretized *heatports*. The thermal connections via the *heatports* represent the interface to the building model. Due to the discretization of the pipe element and, consequently, the floor, each room needs to have the same number of *heatports* corresponding to the discretization number *dis*. Care should be taken to assign the ports in such a way that the floor

Table 1. Assignment of the thermal resistances from the records of the floor layers to the model on room level

Floor Layer	Position in Record	Thermal Resistance
Flooring	wallTypeFloor[2]	$R_{\lambda,\mathrm{B}}$
Screed	wallTypeFloor[1]	$s_\mathrm{u}/\lambda_\mathrm{E}$
Insulation	wallTypeCeiling[1]	$R_{\lambda,\mathrm{Ins}}$
Load-bearing substrate	wallTypeCeiling[2]	$R_{\lambda,\mathrm{Ceiling}}$
Plaster	wallTypeCeiling[3]	$R_{\lambda,\mathrm{Plaster}}$
Foam glas in floor plate	wallTypeCeiling[3]	$R_{\lambda,\mathrm{Ceiling}}$
Gravel under floor plate	wallTypeCeiling[4]	$R_{\lambda,\mathrm{Ceiling}}$

heating is located between the wall models for floor and ceiling.

The design according to EN 1264-3 (2020) is done in the model with the room that is connected to the building model first. Since the design is done with the room that has the highest specific heating load, care must be taken to ensure that this room is first in the vectorial setup. If this is not the case, the model will show an error. It is therefore essential that the building model is subjected to a heat load calculation before being used with the model for the underfloor heating system. The final design of the entire underfloor heating system according to EN 1264-3 (2020) determines a general flow temperature for the system and the nominal mass flow rate for each room.

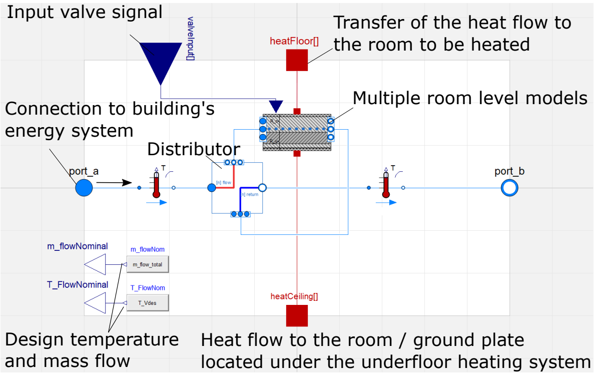

Input valve signal
valveInput[]
heatFloor[]
Transfer of the heat flow to the room to be heated
Multiple room level models
Connection to building's energy system
Distributor
port_a
T
[n] flow
[n] return
T
port_b
m_flowNominal
m_flowNom
m_flow_total
T_FlowNominal
T_FlowNom
T_Vdes
heatCeiling[]
Design temperature and mass flow
Heat flow to the room / ground plate located under the underfloor heating system

Figure 5. Model *UnderfloorHeatingSystem* in which the needed underfloor heating circuits are generated and designed for each heated room

Figure 5 depicts a schematic of the *UnderfloorHeatingSystem* which unifies the heating circuits of the individual heated rooms. It includes the previously presented models of chapters 2.1-2.3 in a hierarchical structure. It contains an automated design according to EN 1264-3 (2020) and the nominal pressure loss calculation of the heating circuit valves. Additional heating circuit parameters can be assigned to each room via the vector parameterization.

3 Results and Discussion

The following chapter is divided into two parts. Firstly, the model will be verified in a simplified test scenario. This is followed by a parameter study to investigate the influences of various system parameters such as the flooring or the pipe spacing.

3.1 Model verification

The verification of the model is performed with two in a simplified way designed room models. The first room has a specific heat load of $q''_{\mathrm{des},1} = 50\,\mathrm{W\,m^{-2}}$ while the second room's heat load is $q''_{\mathrm{des},2} = 33\,\mathrm{W\,m^{-2}}$. The heat transfer from the floor surface to the room is mapped according to EN 1264-3 (2020) with a purely convective heat transfer coefficient of $\alpha_\mathrm{floor} = 10.8\,\mathrm{W\,m^{-2}\,K^{-1}}$. The verification is carried out based on the room and return temperatures that occur. The expected values must be achieved for various floor heating parameters within the standard limits. For this reason, the simulation is performed with different parameters for pipe spacing, inner diameter and pipe material. The model is verified using the target room temperature of 20 °C and the calculated return temperature from the standard. In summary, the model was verified under the following aspects:

- Room parameters (room size, specific heat load, floor/ceiling wall parameters) of design room transferred into the underfloor system model
- Flow temperature and mass flow are equal to design temperature and design mass flow, which are calculated by the model itself (Using EN 1264-3 (2020))
- There is no further control of mass flow or temperature
- All boundary conditions are steady-state
- $\Delta T_\mathrm{Room} = T_\mathrm{design} (= 20\,°\mathrm{C}) - T_\mathrm{Room}$
- $\Delta T_\mathrm{r} = \vartheta_\mathrm{design} - \vartheta_\mathrm{r}$

Table 2 shows the results of the room ΔT_Room and return temperatures $\vartheta_\mathrm{r}, \Delta T_\mathrm{r}$ from the simulations for verification as absolute values with deviation.

The results show that the designed room with the higher heating load has room temperatures between $\vartheta_\mathrm{room1} = 19.9892 \ldots 19.9935$ °C which is slightly below the target room temperature of 20 °C The return temperature

Table 2. Results of the verification of the overall model: Consideration of two rooms with different heating loads at different underfloor heating parameters

Room 1: $q''_{\mathrm{des},1} = 50\,\mathrm{W\,m^{-2}}$

	$\Delta T_{\mathrm{Room}}[\mathrm{K}]$	$\vartheta_{\mathrm{r}}[°\mathrm{C}]$	$\Delta T_{\mathrm{r}}[\mathrm{K}]$
T = 100 mm	-0.0108	31.7458	-0.0011
T = 200 mm	-0.0065	34.7029	-0.0010
T = 350 mm	-0.0075	40.2079	-0.0010
d_{i} = 21 mm	-0.0071	38.2973	-0.0010
PP-pipe	-0.0069	41.5902	-0.0010

Room 2: $q''_{\mathrm{des},2} = 33\,\mathrm{W\,m^{-2}}$

	$\Delta T_{\mathrm{Room}}[\mathrm{K}]$	$\vartheta_{\mathrm{r}}[°\mathrm{C}]$	$\Delta T_{\mathrm{r}}[\mathrm{K}]$
T = 100 mm	+0.0105	24.6451	-0.0024
T = 200 mm	+0.0168	25.8692	-0.0027
T = 350 mm	+0.0177	28.1566	-0.0032
d_{i} = 21 mm	+0.0132	27.3619	-0.0030
PP-pipe	+0.0125	28.7318	-0.0033

is achieved to two digits after the decimal point. For the designed room with the lower heating load, the underfloor heating system calculates room temperatures between $\vartheta_{\mathrm{room2}}$ = 20.0105...20.0132 °C which are thus slightly above the set temperature. The mean absolute percentage error (MAPE) for the return temperature of the underfloor heating system is less than 0.011 %. The room temperature is achieved with a MAPE of 0.0388 % for the designed room 1 and 0.0706 % for the designed room 2.

Thus, the model can be seen as verified for the system requirements according to EN 1264-2 (2021). The small deviations are due to the division of heat transfer at the floor surface into radiation and convection. While the design according to the standard assumes constant heat transfer coefficients, the model considers the dependence of radiation on the room's surface temperature. This justifies small deviations in the heat transfer.

3.2 Influence of system parameters

A parameter study is performed to investigate the influence of the variable parameters on the the underfloor heating system. The influences of pipe spacing T, pipe materials with different thermal conductivities $\lambda_{\mathrm{material}}$, pipe diameter d_{i} and pipe coating s_{C} are considered. Besides, the changes due to a variation of floor layers with different thermal resistances $R_{\lambda,\mathrm{F}}$, screed thicknesses s_{u} and insulation thicknesses s_{ins} are investigated. The results for supply and return temperatures are examined since these are of greatest importance for a building energy system. For this purpose, all simulations are performed with a discretization of 50 pipe elements. The floors of all rooms are discretized as well. The parameter study is performed in conjunction with the single-family house of the High Order Model from the AixLib library (Constantin, Streblow, and Müller 2014). Table 3 shows the range of variation of the parameters and the reference parameters.

Table 3. Reference and range of variation of different system parameters

parameter		reference	min.	max.
T	[mm]	200	100	400
d_{i}	[mm]	13	10	20
$\lambda_{\mathrm{material}}$	$[\mathrm{W\,m^{-1}\,K^{-1}}]$	PE-RT	0.22	390
s_{C}	[mm]	0	0	4
$R_{\lambda,\mathrm{F}}$	$[\mathrm{m\,K\,W^{-1}}]$	0.1	0.0118	0.188
s_{u}	[mm]	60	40	80
s_{ins}	[mm]	35	35	75

Generally, decreases in the total heat transfer within the system are compensated by an increase in the flow temperature. If the heat transfer to the room is not directly affected, changes are compensated by increasing the mass flow. This is to ensure the required heat flow at all times.

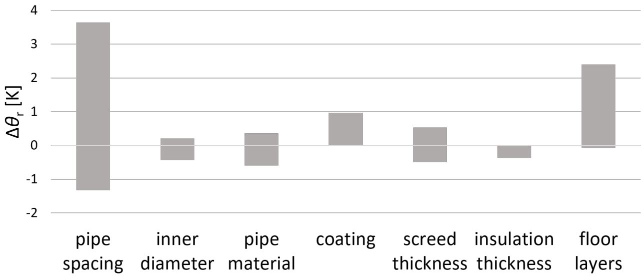


Figure 6. Influence of different system parameters on the return temperature

Figure 6 summarizes the changes in return temperature due to various system adjustments. The pipe spacing has the greatest influence on the temperature level of the system. The return temperature can increase up to 3.5 K due to the necessary increase in the supply temperature. Also, the floor layer has a non-negligible influence. Thus, the return temperature varies by about 2.5 K for the used floor layers (elastomer, laminate, wood parquet, carpet, linoleum). The variation of the pipe material, the pipe sheathing and the thickness of the screed layer are of little significance concerning the return temperature. The return temperature changes by less than 1 K. The thickness of the insulation layer also has the least effect on the return temperature. Primarily, the changes in heat transfer due to the insulation are compensated with an adjustment of the mass flow.

4 Conclusion and Outlook

In this work, we present an underfloor heating system model, which is built on four submodels. The model is freely available on GitHub:
https://github.com/RWTH-EBC/AixLib/tree/issue890_HOMProject_FloorHeating.
The smallest unit is a pipe element, which is connected

in a row for several times to form an axial discretized heating circuit.

Based on this, the next level model forms a connection for a room, which may consist of several heating circuits. The overall model connects several rooms to form an underfloor heating system, whose heating circuits converge in a distributor and enable connection for a building energy system.

The specifications of EN 1264-3 (2020) are included in the model to allow direct use of the calculated supply temperature and design mass flow. The model is verified for the system requirements of EN 1264-2 (2021) and shows an accuracy of less than 0.01 % (MAPE) concerning the room temperature. In a parameter study, the influence of different system parameters is investigated. The pipe spacing has the greatest influence and can cause an increase in the flow temperature by up to 10.9 K.

All in all, we show an underfloor heating model in Modelica which is automatically parameterized by the integrated variables of the standard. In future work, the model will be integrated into building energy systems to allow a holistic analysis of those.

Acknowledgements

This work has been supported by Federal Ministry for Economic Affairs and Energy (BMWi), (grant numbers 03ET1595A).

References

Borrajo Bastero, Josué et al. (2019). “Model and Validation of a Multi-family Building ‘Haus M’ Using Modelica”. In: Building Simulation 2019. Rome, Italy, pp. 4235–4242. DOI: 10.26868/25222708.2019.210763. URL: http://www.ibpsa.org/proceedings/BS2019/BS2019_210763.pdf.

Constantin, Ana, Rita Streblow, and Dirk Müller (2014). “The Modelica HouseModels Library: Presentation and Evaluation of a Room Model with the ASHRAE Standard 140”. In: *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, pp. 293–299. DOI: 10.3384/ECP14096293.

EN 1264-1 (2021). *Water based surface embedded heating and cooling systems - Part 1: Definitions and symbols*. European Committee for Standardization.

EN 1264-2 (2021). *Water based surface embedded heating and cooling systems - Part 2: Floor heating: Methods for the determination of the thermal output using calculations and experimental tests*. European Committee for Standardization.

EN 1264-3 (2020). *Water based surface embedded heating and cooling systems - Part 3: Dimensioning*. Berlin: European Committee for Standardization.

EN 1264-4 (2021). *Water based surface embedded heating and cooling systems - Part 4: Installation*. Berlin: European Committee for Standardization.

Hestermann, Ulf et al. (2010). *Baukonstruktionslehre 1: Mit 138 Tabellen*. 35., vollständig überarbeitete und aktualisierte Auflage. Wiesbaden: Vieweg+Teubner Verlag / GWV Fachverlage GmbH Wiesbaden. ISBN: 978-3-8348-0837-0. DOI: 10.1007/978-3-8348-9386-4. URL: http://dx.doi.org/10.1007/978-3-8348-9386-4.

Müller, Dirk et al. (2016). “AixLib - An Open-Source Modelica Library within the IEA-EBC Annex60 Framework”. In: *CESBP Central European Symposium on Building Physics/BauSIM 2016*. Ed. by John Grunewald. Stuttgart: Fraunhofer IRB Verlag, pp. 3–9. URL: urn:nbn:de:101:1-201612202736.

Schütz Energy Systems (2017). *Heizkreisverteiler: Montageanleitung/- Technische Information*. URL: https://www.schuetz-energy.net/downloads/anleitungen/montageanleitung-heizkreisverteiler/schuetz-montageanleitung-fbh-heizkreisverteiler-de.pdf?cid=4jt.

Taschenbuch für Heizung + Klimatechnik 13/14 (2012). 76. Aufl. ISBN: 3-8356-3301-5.

Weitzmann, Peter et al. (2005). “Modelling floor heating systems using a validated two-dimensional ground-coupled numerical model”. In: *Building and Environment* 40.2, pp. 153–163. ISSN: 03601323. DOI: 10.1016/j.buildenv.2004.07.010. URL: https://linkinghub.elsevier.com/retrieve/pii/S0360132304001702.

ScalableTestGrids - An Open-Source and Flexible Benchmark Suite to Assess Modelica Tool Performance on Large-Scale Power System Test Cases

Francesco Casella[1] Adrien Guironnet[2]

[1]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy,
francesco.casella@polimi.it
[2]Réseau de Transport d'Electricité, France, adrien.guironnet@rte-france.com

Abstract

This paper introduces *ScalableTestGrids*, an open-source and flexible benchmark suite for assessing the performance of Modelica tools on large-scale power system test cases. The benchmark suite is built by using components from the PowerGrids library and generic utility scripts creating the final runnable Modelica models. It does not depend on confidential data; its structure makes any future needed modification or evolution easy and straightforward. Results obtained with the OpenModelica tool are also reported. The benchmark suite can be used by tool developers to assess the capability of their tools to handle large-scale power generation and transmission models.

Keywords: Benchmark, Power System Simulation, Large-Scale Simulation, Performance, Open-Source

1 Introduction

Power systems are evolving at a very fast pace due to a global demand for cleaner energy. This drives major changes in the system structure, with a growing penetration of Renewable Energy Sources (RES) and an important boom of High-Voltage Direct Current (HVDC) lines (ENTSO-E 2020; IEA 2021). To be able to ensure the system stability and to handle the new challenges arising in a satisfying manner, System Operators (SO) have to adapt the way they control, operate and design the grid. This notably implies that power system simulation tools should offer a great flexibility to cope with the pace of evolution and a high-level of transparency to facilitate collaboration and coordination between all the actors.

While traditional power system software use closed-source models, solvers and data, and are difficult to adapt to emerging technologies, Modelica offers an appealing alternative. Its open-source, declarative and high-level nature makes it a good candidate for modern, flexible and transparent power system modelling.

These advantages have boosted the development of several power system Modelica libraries in the recent years (Winkler 2017), from first efforts to port existing tool models (Bogodorova et al. 2013) to libraries carefully designed to take full advantage of the declarative modelling approach of the language while easing the transition for power system experts who are Modelica beginners (A. Bartolini, F. Casella, and Guironnet 2019). It has also led to academics usage of Modelica in other domains than classical electromechanical simulations, such as electromagnetic transient simulations (Masoom et al. 2020), dynamic phasor modeling (Mirz et al. 2019) or power-electronics dominated grid simulations (Cossart et al. 2020) as well as proof of concept for industrial use (Francesco Casella, Andrea Bartolini, et al. 2016; Guironnet et al. 2018).

At the same time, efforts have been made to ease a wide adoption by both development of automatic conversion methods for creation of standard test cases (Razik, Dinkelbach, et al. 2018; Gómez et al. 2019) and computation time improvements (Francesco Casella, Leva, and Andrea Bartolini 2017; Braun, Francesco Casella, and Bachmann 2017; Henningsson, Olsson, and Vanfretti 2019), for example through the use of DAE-mode integration. However, fundamental performance barriers remain on these two fronts, hampering the full operational use of general-purpose Modelica tools to handle simulations involving national- or continental-scale grids; this motivated for instance the development of a mixed C++/Modelica approach by RTE (Guironnet et al. 2018).

Indeed, it is important to remind the reader the operational constraints currently existing on automation and performance for time-domain simulations in power systems. One fundamental process to ensure power system stability is the so-called *Dynamic Security Assessment*, that consists in simulating a large number of contingencies to make sure that the grid is operating in a secure and stable way (Loud et al. 2010; Panciatici, Bareux, and Wehenkel 2012). For example, the French national grid control center launches 65 different simulation scenarios every 15 minutes for voltage stability, and 1270 different scenarios every 30 minutes for transient stability. In addition to this *real-time* use, similar calculations are done in day-ahead and week-ahead situations with comparable performance constraints.

Currently available general-purpose Modelica simulation tools are still not able to provide adequate support for

DOI
10.3384/ecp21181351

such applications, in particular because the standard approach followed for code generation requires flattening the models to scalar equations, which leads to unacceptable code generation time and generated code size, see, e.g. (Francesco Casella, Leva, and Andrea Bartolini 2017). A quantum leap is required in code generation technology, which should avoid as much as possible to generate repeated code when hundreds or thousands of similar components are instantiated in a system model, as is commonplace in national grid models. Significant improvements may also be required on the simulation runtimes, e.g. to handle events efficiently.

On the other hand, testing the performance of such advanced or experimental Modelica tools on large-scale, realistic national power grid models is not a straightforward task; in general, it requires a lot of domain-specific knowledge to set up realistic test models and meaningful simulation scenarios, and possibly relies on confidential or restricted-access data. The latter issue also makes it difficult to compare the performance of different tools, since different tool developers may not have access to the same models.

In order to fill this gap, the development of a Modelica library based on the PowerGrids library (A. Bartolini, F. Casella, and Guironnet 2019) was launched, to offer open-source, easy-to-use, customizable and scalable benchmarks to all Modelica tools providers. The library captures the essential features of large-scale electromechanical power grid models, while keeping the complexity at the minimum possible level.

This library, called *ScalableTestGrids*, can be used by people doing research and development on advanced methods to handle large-scale Modelica models, to test the ability of their methods and tools to handle the kind of models that are required by SOs to run their daily operation. It can also be used to compare tools against each other, to assess the improvements of a given tool over time, and ultimately to make educated guesses possible about when general-purpose Modelica tools could eventually be used for industrial-grade power system simulations.

The rest of the paper will be organized in the following way. Section 2 presents the requirements used for specifying the benchmark suite while Section 3 introduces the selected design. Section 4 provides the current benchmark suite and gives the results obtained with OpenModelica while sketching evolutions that could further improve the performance. Finally, Section 5 serves as the conclusion.

2 Requirements

This section will introduce the main requirements used to define the benchmark suite and their motivations.

The first requirement is that the benchmark suite is representative of real power system test cases. This point is very important and ensures that advanced tools take advantage of structural conditions really existing in large-scale networks, rather than exploit artificial structural conditions only appearing in the benchmark suite. In particular, large-scale power system test cases have two main characteristics.

The first one is the very sparse structure of power system. Indeed, in power systems, each electrical node is only connected to a few other ones and the other physical components are either connections between two nodes or are interfaced to a single node. Controls are essentially local even if a few wide-area ones exist but only affect a very restricted subset of variables (frequency regulation for example). Table 1 shows representative sparsity levels for large-scale networks and Figure 1 shows the French power system and enables to directly see the sparse nature of the grid.

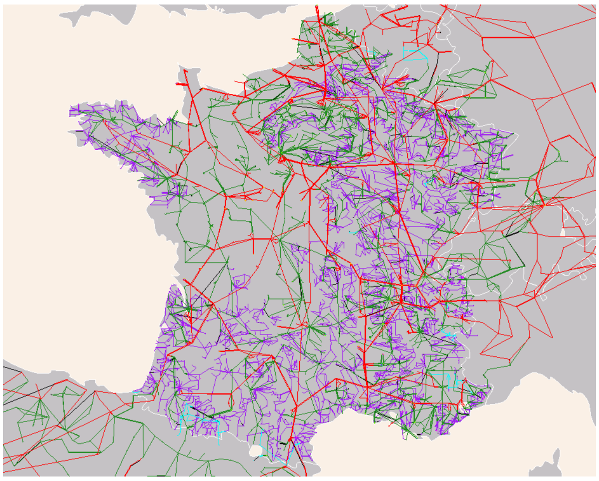


Figure 1. French Transmission System

The second characteristic is that large-scale networks are built using relatively few component models, which are instantiated a large number of times. This feature should be exploited by the tool, which should avoid wasting time and memory to generate repeated code structures. The components are instantiated and connected on a one-by-one basis, coming from a netlist description of the grid structure, which is irregular, as shown in Figure 1. Table 2 illustrates this property with the number of components for different large-scale test cases; the figures are taken from the Horizon 2020 Pegase project reports.

The third requirement for the benchmark suite is that it should be easily modified. In its current version the benchmark suite focuses on the impact of the system size, particularly the number of continuous variables, on the tool performance. This already offers a lot of tough challenges to address, but it could further evolve, for instance taking into account issues arising with large numbers of discrete variables and events. It is definitely of prime importance that such changes can be included in a straightforward way, considering the current pace of evolution in power systems, ensuring that the benchmark suite consistently captures the challenges posed to Modelica tools by such

Table 1. Sparsity levels for representative test cases with K nodes, N equations, NNZ non-zero elements in the Jacobian, density factor $d = \frac{NNZ}{N.N}$ (Razik, Schumacher, et al. 2019)

Test case	**K**	**N**	**NNZ**	**d [%]**
French Extra High Voltage with Simplified Loads	2000	26432	92718	**0.013**
French Extra High Voltage with Voltage-Dependent Loads	2000	60236	188666	**0.0051**
French + one neighbor Extra High Voltage with Simplified Loads	3000	47900	205663	**0.0089**
French + one neighbor Extra High Voltage with Voltage Dependent Load	3000	75300	266958	**0.0047**
French + neighb. countries Extra High Voltage with Simplified Loads	7500	70434	267116	**0.0054**
French Extra High Voltage + regional High Voltage with Simplified Loads	4000	90940	316280	**0.0038**
French Extra High Voltage + regional High Voltage with Voltage Dependant Loads	4000	197288	586745	**0.0015**
French + neighb. countries Extra High Voltage with Voltage Dependent Loads	7500	220828	693442	**0.0014**

Table 2. Number of components instances for representative test cases

Test case	**Generators**	**Loads**	**Lines**	**Transformers**
French EHV	607	2905	2668	1040
French EHV + HV	725	7875	8592	2577
Continental European EHV	3483	7211	~ 16000	~ 5000

models.

The fourth requirement concerns the benchmark suite scalability. The goal being to assess tool performance on large-scale simulations, creating examples of different size in a simple way is a *must have*. It means that from a small test case, utility functions should exist to obtain larger test cases with similar properties. These functions should be as generic as possible to facilitate their use for any kind of test case and should contain parameters enabling to define the final test case (its size for example).

Finally, the fifth requirement considered is related to the usability of the benchmark suite. In order to maximize its potential use, it is necessary that the initialization and simulation work fine in different Modelica environments. On the one hand, this implies that initialization should be straightforward and robust. On the other hand, simulation scenarios which are relevant but straightforward to simulate should be defined, ensuring that there are no potential simulation issues even with the large-sized systems; at the same time, the test cases should remain well representative of real-life scenarios at all sizes. The goal of these benchmarks is to show that a tool can handle systems of realistic size with good performance, not to test corner cases or numerically challenging situations.

3 Design

3.1 Package Structure

Considering the requirements laid out in Section 2, the library has been structured in three main packages:

- A *Components* package, which defines the components (based on the PowerGrids library) used to assemble the test cases.

- A *GridModelGenerators* package that contains the utilities functions to create the actual test cases.

- A *Models* package, containing some instances of automatically generated test cases for convenience, e.g. to get them easily run by continuous integration frameworks.

In the *GridModelGenerators* package, Modelica functions are employed to create large-scale network model using for-loops. This ensures that the library is fully self-contained and does not rely on other languages (e.g. Python or C) to generate the code of the test cases.

3.2 Model structure

A key feature of these models is that components and connect statements are instantiated individually, as in a real-life cases such as the one shown in Figure 1. Relying on arrays of components and for-loop connection equations could generate system structures that could be further optimized by smart tools, but that would be irrelevant to assess the performance on real-life models, where such regular structures are absent.

The structure of the scalable grid model which is currently implemented as a benchmark in the library is shown in Figure 2. At the top level, a meshed grid is represented, includings some nodes with large power generation systems, and some nodes with connection to local radial grids, which have a linear or tree-like structure. Twin transmission lines are sometimes used to ensure higher transmission capacity in the meshed part.

These structural features were represented by an idealized square, $2N \times 2N$-node meshed grid, with alternating generation (round) and load (square) nodes. The nodes are connected in the east-west direction by means of single transmission lines, while twin parallel transmission lines are used in the north-south direction.

Generator nodes contain a full-fledged synchronous machine model, equipped with automatic voltage regulation (IEEE AC4A type), power system stabilizer (IEEE PSS2A type), and turbine governor with primary frequency control (IEEE TGOV type), built with components from the PowerGrids library. The generator is connected to the meshed grid by a step-up transformer.

Load nodes contain a sub-system, built by the linear connection of M transmission lines, each connected to a PQ load at its end, and ultimately connected to the meshed network by a step-up transformer.

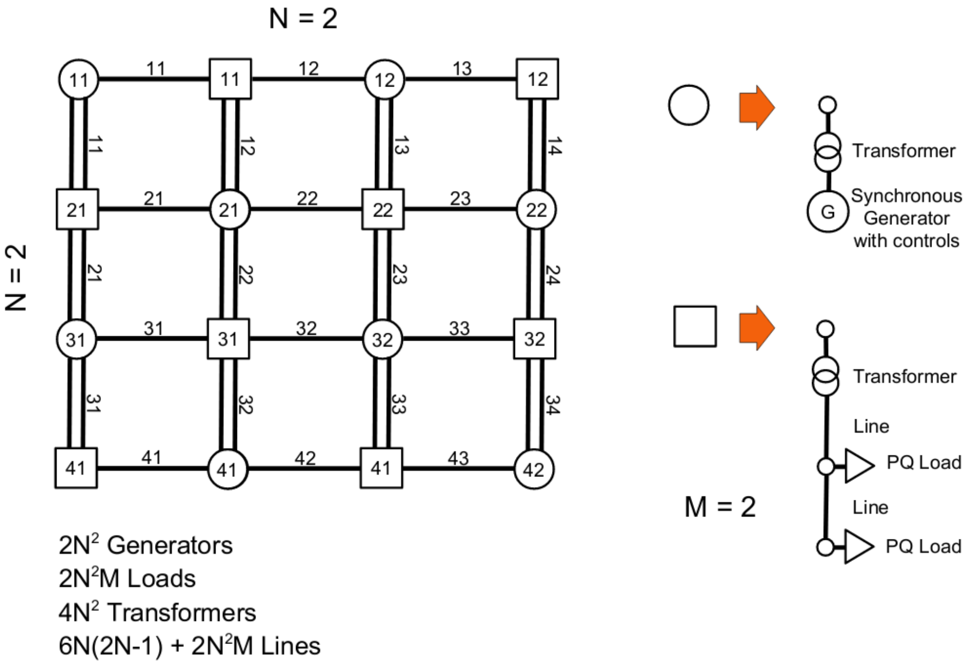


Figure 2. Test case structure with N = M = 2

As one can notice on the figure, the final test case comprises $2N^2$ generators, $2N^2M$ loads, $4N^2$ transformers and $6N(2N-1)+2N^2M$ lines. Thus, by setting N one can change the overall system size, while changing M allows to modify the sparsity pattern and include a more or less important part of the radial network in the system model.

The spatial structure of the grid is somewhat idealized, having a very regular pattern that can be easily scaled up in size. On the other hand, individual components are instantiated and connected one-by-one, and full-fledged realistic component models are used, so that the performance of a Modelica tool on a benchmark suite containing the number of nodes listed in Figure 2 can be considered to be fully representative of the performance on a model of a real-life system of the same size.

3.3 Initialization and simulation

Regarding initialization, dynamic power grid models, including the ones built with the *PowerGrids* library, are strongly nonlinear, and thus require the results of a static power flow computation to set up the start values for initialization; this is normally taken from the output of a separate tool. However, this would really be inconvenient in the case of this benchmark suite. The test model was thus conceived in order to have an initial power flow that is easily computed by the generated Modelica model, exploiting symmetry features.

Consider first an idealized case, where an infinite number of nodes is present, with full symmetry of voltages and power flows. Each generator node would then be surrounded by four load nodes of identical voltage, while each load node would be surrounded by four generator nodes of identical voltage. Also, there is zero net active power exchange between pairs of synchronous generators, which thus have all the same phase. Assuming that the load nodes have their nominal voltage magnitude (1 p.u.), and that the generator node voltages have a zero phase, it is possible to compute the complex power flows through the lines surrounding each load node, and hence the power flows and voltage phase and magnitude at both generator and load nodes. The active power output of each generator is then set to be equal to the total active power input of each radial sub-system, where each load takes $1/M$-th of the total active load, plus a small extra term for resistive losses across the transmission lines.

In this way, power flow values can be easily computed analytically and the results can be used to directly set the start values of voltage and complex power at the generator and load ports. The actual values of this ideal power flow are thus hard-wired in the code generator of the system model.

In fact, the actual grid has a finite extent, so it shows some border effects, compared to the ideal symmetric infinite grid; for example, the generator at node 11 needs to send its power through three lines only, instead of six, thus it requires a somewhat higher voltage. However, the difference with respect to the symmetric case is small enough that the symmetric power flow results can be used as start values for the finite grid steady-state initialization problem, without causing any convergence issue.

The only requirements on the Modelica tool used to simulate the test case are that it should use a *sparse* nonlinear solver to handle the initialization problem, since its size prevents using a dense solver in all cases except the smaller ones, and that it should preferably use homotopy for the initialization process, to avoid issues related to the controller saturations.

Once the system has been initialized at steady state, the simulation scenario assumes that all loads in the upper half of the network reduce their active power consumption by 10% at $t = 1$, regardless of the system size. This starts a transient in which all synchronous generators initially accelerate, due to the overall power imbalance; then, the primary frequency controllers reduce the turbine power outputs and stabilize the grid at a slightly higher frequency. Some local and inter-area damped oscillations of voltage and frequency ensue.

The symmetric nature of this perturbation is such that the transients of individual currents and power flows in the grid components in certain areas of the grid are similar regardless of the system size. This makes the comparison of simulation times across different grid sizes fair, since more or less the same things happen in each component, regardless of the grid size. This would not be the case if, e.g., the perturbation was applied to one load only, since the relative effect of such a perturbation would become smaller with increasing system size, potentially requiring less time steps from the variable step-size solver.

3.4 Generation of system models

Figure 3 shows a code fragment of the Modelica function that generates the system models, showing how individual components are instantiated and how individual connect-equations are added to the system.

The *Components* package contains the component models that are instantiated by the system model. In the current library version, as already mentioned, the generator model utilized is the connection of a synchronous generator with a voltage regulator control (IEEE AC4A), a power system stabilizer (IEEE PSS2A) and a speed regulator (IEEE TGOV). All these models are connected together to create the *ControlledGenerator* model that is instantiated in the system model.

This model structure makes it easy to modify or extend the test case, to include a new model or new levels of complexity. For example:

- in order to assess the impact of discrete variables and events, it is possible to create a composite model using the transformer physical model and a tap-changer logic (available in the original PowerGrids library) in the *Components* package, using it in place of the simple transformer physical model when instantiating the radial part of the network;

- if one wants to see the effect of distributed RES on performances, the load nodes can be enriched by connecting a RES model in parallel with the PQ loads;

- to include a new structure in the test cases, such as a few HVDC lines between two sub-networks, an extra for-loop could be included in the code generation function.

All these variants could be handled by Boolean parameters of the code generation function, which would activate the generation of the corresponding code.

4 Benchmark suite and results

The library has been used to create a first set of test cases using the basic model described in Section 3.2: synchronous generators with frequency and voltage regulations, voltage-dependent loads, classical linear transmission lines and transformers models (no regulations nor saturations).

Test cases were run with different values of N and M using OpenModelica version 1.18.0-dev-263-g3806526c07, setting the the most favourable options: use of *DAE-mode*, fixed-step homotopy solver for the initialization part, sparse Kinsol/KLU solver for the time-domain part, no tearing, which is too cumbersome for large systems, and -O0 optimization of the C code compilation, to avoid spending too much time on executable code optimizations.

The same experiments were performed on two different machines: a 20-core Xeon E5-2650 workstation with 72 GB of RAM under Ubuntu 20.04, and on an 8-core i7-8550U laptop with 16 GB of RAM under Windows 10 Pro, both 64-bits. During code generation, OpenModelica can exploit multiple cores by running several threads in parallel, e.g. for garbage collection; the generated C code is also split in several files, that can be compiled in parallel. Hence, simulations were run one at a time, to exploit parallelism a much as possible.

The results obtained are collected in Table 3. Simulations were ran successfully up to N = 11, M = 4 on the workstation and up to N = 6, M = 4 on the laptop; larger test cases could not be run due to memory limitations. However, the expected performance figures for those cases were extrapolated based on smaller test case results, and shown in italics in the table.

Concerning the simulation part, performance remain acceptable up to about 4000 components (i.e., generators, transformers, lines, and loads) in the system. It is comparable to the performance of existing domain-specific tools and demonstrates that the computation time is compatible with industrial uses of Modelica-based solutions for power system stability. Above this size, further improvements are needed, both on the software side - symbolic Jacobians in *DAE-mode* for simulation, more efficient and streamlined run-time code - and on the hardware side - using last-generation high-performance hardware, for example.

Code generation and compilation have reasonable performance up to about 300 components in the system. This means that general-purpose Modelica-based tools such as OpenModelica can be used for research studies on small

```
algorithm
  when initial() then
    Modelica.Utilities.Files.remove(f);
    print("within ScalableTestGrids.Models;", f);
    print("model Type1_N_" + String(N) + "_M_" + String(M), f);
    print("  extends Modelica.Icons.Example;", f);
    print("  inner PowerGrids.Electrical.System systemPowerGrids(", f);
    print("    initOpt = PowerGrids.Types.Choices.InitializationOption.globalSteadyStateFixedPowerFlow);", f);
    for i in 1:2 * N loop
      for j in 1:N loop
        if i == N and j == div(N + 1, 2) then
          print("  PowerGrids.Electrical.Buses.ReferenceBus BUS_GEN_EHV_" + String(i) + "_" + String(j) + "(SNom = 1e9, UNom = 400e3,
               UStart = 400e3 * 0.966, portVariablesPhases = true);", f);
        else
          print("  PowerGrids.Electrical.Buses.Bus BUS_GEN_EHV_" + String(i) + "_" + String(j) + "(SNom = 1e9, UNom = 400e3,
               portVariablesPhases = true);", f);
        end if;
      end for;
    end for;
    for i in 1:2 * N loop
      for j in 1:N loop
        print("  PowerGrids.Electrical.Buses.Bus BUS_LOAD_EHV_" + String(i) + "_" + String(j) + "(SNom = 1e9, UNom = 400e3,
             portVariablesPhases = true);", f);
      end for;
    end for;
    for i in 1:2 * N loop
      for j in 1:N loop
        print("  Components.ControlledGenerator GEN_" + String(i) + "_" + String(j) + "(GEN(SNom = 1e9, PStart = -806e6, QStart = -300
             e6));", f);
      end for;
    end for;
    for i in 1:N loop
      for j in 1:N loop
        for k in 1:M loop
          print("  PowerGrids.Electrical.Loads.LoadPQVoltageDependence LOAD_" + String(i) + "_" + String(j) + "_" + String(k) + "(PRef
               = Pvar, QRef = Qvar, UNom = 63e3, SNom = " + String(1e9 / M) + ", PStart = " + String(800e6 / M) + ", QStart = " +
               String(100e6 / M) + ");", f);
        end for;
      end for;
    end for;

    ...

    print("equation", f);
    for i in 1:2 * N loop
      for j in 1:N loop
        print("  connect(BUS_GEN_EHV_" + String(i) + "_" + String(j) + ".terminal, TRANSFORMER_GEN_" + String(i) + "_" + String(j) + "
             .terminalB);", f);
      end for;
    end for;
    for i in 1:2 * N loop
      for j in 1:N loop
        print("  connect(GEN_" + String(i) + "_" + String(j) + ".terminal, TRANSFORMER_GEN_" + String(i) + "_" + String(j) + ".
             terminalA);", f);
      end for;
    end for;

    ...
```

Figure 3. Part of the algorithm implemented to create the Modelica code of the system models

Table 3. Performance results for different system sizes

										Xeon E5-2650, Ubuntu 20.04				i7 85506, Windows 10			
N	M	# equations	# non-trivial eqs.	# nodes	# generators	# transformers	# lines	# loads	# solver steps	code gen. time / s	C compile time / s	exec size / MB	sim. time / s	code gen. time / s	C compile time / s	executable size / MB	simulation time / s
2	4	12174	5091	80	8	16	68	32	297	17.4	3.6	13.5	0.8	26.7	12.7	21.0	1.0
3	4	28284	11801	180	18	36	162	72	319	40.7	8.9	31.2	1.9	66.8	25.1	36.7	2.2
4	4	51078	21307	320	32	64	296	128	315	75.3	15.7	56.2	3.5	125.3	41.2	59.0	4.2
6	4	116718	48683	720	72	144	684	288	294	178.5	36.8	128.2	8.2	305.4	66.0	123.1	10.7
8	4	209094	87211	1280	128	256	1232	512	300	329.1	69.6	229.6	15.8	*600.0*	*140.0*	*240.0*	*20.0*
11	4	397788	165913	2420	242	484	2354	968	322	737.5	163.8	438.0	34.6	*1200.0*	*300.0*	*500.0*	*40.0*
16	4	*800000*	*350000*	5120	512	1024	5024	2048	*300*	*1500.0*	*350.0*	*900.0*	*70.0*	*2400.0*	*600.0*	*1000.0*	*80.0*
23	4	*1600000*	*660000*	10580	1058	2116	10442	4232	*300*	*3000.0*	*800.0*	*1800.0*	*150.0*	*5000.0*	*1200.0*	*2000.0*	*160.0*
32	4	*3200000*	*1400000*	20480	2048	4096	20288	8192	*300*	*6000.0*	*1800.0*	*3600.0*	*320.0*	*10000.0*	*2500.0*	*4000.0*	*320.0*
45	4	*6400000*	*2600000*	40500	4050	8100	40230	16200	*300*	*12000.0*	*4000.0*	*7500.0*	*700.0*	*20000.0*	*5000.0*	*8000.0*	*640.0*
64	4	*12800000*	*5600000*	81920	8192	16384	81536	32768	*300*	*24000.0*	*9000.0*	*18000.0*	*1500.0*	*40000.0*	*10000.0*	*16000.0*	*1280.0*

test cases without any problem; as soon as the number of components increases more, code generation becomes too time-consuming for practical industrial use. As already mentioned in the Introduction, as long as the structural analysis and symbolic processing of the system equations is done on a scalar basis, the code generation process and the generated code size do not scale up well enough.

The use of non-expanded arrays in all the stages of the compilation of the Modelica code into simulation code, which would exploit the repeated instantiation of the same model a large number of times, seems to be the most promising path to push back these limits. To the author's knowledge, such methods are currently not yet implemented in any production-grade, general-purpose standard Modelica tool. They are in fact already available in the IDA simulation tool (Sahlin et al. 2019), which however supports a variant of the language, IDA Modelica, a subset of the full Modelica language, with some extensions for separate compilation. Unfortunately the authors did not have access to that tool at the time of this writing, so they were unable to assess its performance with the proposed suite of benchmark models.

The results obtained clearly show that the most important barrier for large-scale simulations is currently found in the code generation and compilation time. This is an even more difficult challenge, knowing that state-of-the-art, domain-specific power system simulation tools don't need such phases during their operation and that one key process is stability assessment demanding a large number of simulations. On the other hand, the particular structure of large-scale power system simulations, built by a small number of different components which are instantiated a large number of times in the system, is not yet really exploited in the back-end process, meaning that a large room for improvement exist.

5 Conclusions

This paper introduces the *ScalableTestGrids* benchmark suite, created on top of the *PowerGrids* library. This open-source library, which is hosted on GitHub (*ScalableTestGrids library* 2021), offers a simple way to build benchmark models of electro-mechanical power generation and transmission systems, that are scalable up to very large size, representative of real-life system models, easy to customize, and easy to simulate, except for their sheer size.

Experiments with the latest version of OpenModelica were carried out on the benchmark suite, proving that the simulation times that can be achieved with general-purpose Modelica tools are already satisfactory. On the other hand, results showed that fundamental progress is still needed in the code generation and compilation phases, to envision an industrial use of general-purpose Modelica tools for large-scale power system stability simulations. In particular, a quantum leap in code generation methods is required, to exploit the feature of these system models, that contain very large numbers of instances of relatively few component models.

In the future, the authors plan to use the benchmark suite to measure the improvements achieved in Modelica tools for large-scale simulations, especially through the use of vectorized code generation. Efforts are ongoing in this direction, for example the LargeDyn project at Linkoping University, and HiPerMod project at Politecnico di Milano. They will also continue enriching it – e.g., by introducing more discrete variables or by adding new power system components such as RES or HVDC lines – to be able to assess the tools performance with diverse power system structures.

As a final remark, the comparison of the performance of different Modelica tools on the presented benchmark suite goes beyond the scope of the present paper, since the challenging nature of the benchmark suite may require to use experimental or undocumented features of the tools (which are not know to the authors of this paper), to obtain the best performance, as was the case with OpenModelica. Other Modelica tool developers are thus encouraged to test their tools on this benchmark suite, to report their best results, and to use it as a reference case to improve the support of large power system modelling in their tools.

References

Bartolini, A., F. Casella, and A. Guironnet (2019-02). "Towards Pan-European Power Grid Modelling in Modelica: Design Principles and a Prototype for a Reference Power System Library". In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. Linköing University Electronic Press.

Bogodorova, T. et al. (2013). "A Modelica power-system library for phasor time-domain simulation". In: *Proc. 4th IEEE PES ISGT Europe*.

Braun, Willi, Francesco Casella, and Bernhard Bachmann (2017-05). "Solving large-scale Modelica models: new approaches and experimental results using OpenModelica". In: *Proc. 12th International Modelica Conference*. Prague, Czech Republic, pp. 557–563. DOI: 10.3384/ecp17132557.

Casella, Francesco, Andrea Bartolini, et al. (2016-10). "Object-Oriented Modelling and Simulation of Large-Scale Electrical Power Systems using Modelica: a First Feasibility Study". In: *Proceedings of the 42nd Annual Conference of the IEEE Industrial Electronics Society IECON 2016*. IEEE. Firenze, Italy: IEEE, pp. 0–6. ISBN: 978-1-5090-3474-1.

Casella, Francesco, Alberto Leva, and Andrea Bartolini (2017). "Simulation of large grids in OpenModelica: reflections and perspectives". In: *Proc. 12th International Modelica Conference*. Prague, Czech Republic, pp. 227–233. DOI: 10.3384/ecp17132227.

Cossart, Q. et al. (2020-10). "An Open-Source Implementation of Grid-Forming Converters Using Modelica". In: *2020 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)*. IEEE.

ENTSO-E (2020). *ENTSO-E Research, Development and Innovation Roadmap 2020-2030*. Tech. rep. ENTSO-E. URL: https://eepublicdownloads.azureedge.net/clean-documents/Publications/RDC%20publications/entso-e-rdi_roadmap-2020-2030.pdf.

Gómez, Francisco J. et al. (2019-01). “CIM-2-mod: A CIM to modelica mapping and model-2-model transformation engine”. In: *SoftwareX* 9, pp. 161–167. DOI: 10.1016/j.softx.2019.01.013.

Guironnet, A. et al. (2018-10). “Towards an open-source solution using Modelica for time-domain simulation of power systems”. In: *Proc. 8th IEEE PES ISGT Europe*. Sarajevo, Bosnia and Herzegovina.

Henningsson, E., H. Olsson, and L. Vanfretti (2019-02). “DAE Solvers for Large-Scale Hybrid Models”. In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. Linköing University Electronic Press.

IEA, RTE (2021). *Conditions and Requirements for the Technical Feasibility of a Power System with a High Share of Renewables in France Towards 2050*. Tech. rep. IEA, RTE. URL: https://assets.rte-france.com/prod/public/2021-01/RTE-AIE_rapport%20complet%20ENR%20horizon%202050_EN.pdf.

Loud, Lester et al. (2010-08). “Hydro-Québec’s challenges and experiences in on-line DSA applications”. In: pp. 1–8. DOI: 10.1109/PES.2010.5588120.

Masoom, A. et al. (2020-12). “Simulation of electromagnetic transients with Modelica, accuracy and performance assessment for transmission line models”. In: *Electric Power Systems Research* 189, p. 106799.

Mirz, M. et al. (2019-07). “DPsim—A dynamic phasor real-time simulator for power systems”. In: *SoftwareX* 10, p. 100253.

Panciatici, Patrick, Gabriel Bareux, and Louis Wehenkel (2012-09). “Operating in the Fog: Security Management Under Uncertainty”. In: *Power and Energy Magazine, IEEE* 10, pp. 40–49. DOI: 10.1109/MPE.2012.2205318.

Razik, Lukas, Jan Dinkelbach, et al. (2018-10). “CIMverter–a template-based flexibly extensible open-source converter from CIM to Modelica”. In: *Energy Informatics* 1.S1. DOI: 10.1186/s42162-018-0031-5.

Razik, Lukas, Lennart Schumacher, et al. (2019-06). “A Comparative Analysis of LU Decomposition Methods for Power System Simulations”. In: *Proc. 2019 IEEE PowerTech*. IEEE. Milan, Italy, pp. 1–6. DOI: 10.1109/PTC.2019.8810616.

Sahlin, Per et al. (2019-09). “On the scalability of equation-based building and district simulation models”. In: *Proceedings 16th IBPSA International Conference and Exhibition*. Rome, Italy, pp. 2584–2590. DOI: 10.26868/25222708.2019.210130.

ScalableTestGrids library (2021). URL: https://github.com/PowerGrids/ScalableTestGrids (visited on 2021-05-07).

Winkler, D. (2017-09). “Electrical Power System Modelling in Modelica - Comparing Open-source Library Options”. In: *Proc. 58th SIMS*. Reykjavik, Iceland.

Continuous Development and Management of Credible Modelica Models

Leo Gall[1] Martin Otter[2] Matthias Reiner[2] Matthias Schäfer[1] Jakub Tobolář[2]

[1]LTX Simulation GmbH, Munich, Germany, leo.gall@ltx.de
[2]German Aerospace Center (DLR), Institute of System Dynamics and Control, Wessling, Germany

Abstract

Modeling and simulation is increasingly used in the design process for a wide span of applications. Rising demands and the complexity of modern products also increases the need for models and tools capable to cover areas such as virtual testing, design-space exploration or digital twins, and to provide measures of the quality of the models and the achieved results. In this article, we try to summarize the state-of-the-art and best-practice from the viewpoint of a Modelica language user, based on the experience gained in projects in which Modelica models have been utilized in the design process. Furthermore, missing features and gaps in the used processes are identified.

Keywords: credible model, model requirement, data management, validation, verification, Modelica model

1 Introduction

The modeling of physical systems is a complex process with many decisions, simplifications and assumptions on its way, usually done by many different decision-makers. In the real world, this often leads to simulation models and simulation results that are not well documented. Due to the increasing use of system simulation in nowadays product development, there is an increasing need in traceable model development with guarantees on model validity.

Models are often shared across organizational borders (for example, from supplier to OEM). On this way the direct access to model sources (for example to internal repositories) and the model history is lost. To avoid this, models have to "carry" documentation "with them". If only Black-Box Functional Mock-Up Units (FMU)[1] or result data with simulation reports are shared, the requirements on credibility and traceability are even higher.

The ITEA 3 project UPSIM[2] aims for system simulation credibility via introducing a formal simulation quality management approach, encompassing collaboration and continuous integration for complex systems. It shall be based on the recently proposed "Credible Simulation Process" (Heinkel and Steinkirchner 2021).

In this article, it is tried to summarize the state-of-the art in the development and management of credible Modelica models, as a basis for future improvements in the UPSIM project. The goal is to achieve a well-documented, traceable development process for Modelica based "*credible* digital twins".

The paper is structured as follows. First, a general overview of the modeling and simulation process is given in Section 2. Then, in Section 3, the former is particularly discussed, focusing on requirements, level of details, validation, etc. The management of Modelica libraries and simulation data is finally elaborated in Section 4.

Even if several of the addressed points are valid for any kind of model-based simulation, the focus of this paper is on the Modelica point of view.

2 Modeling process

A modeling process can be roughly divided into the following steps, examined in the corresponding sections:

1. Definition of the requirements, Section 3.1.
2. Definition of the modeling task, Section 3.2
   (based on the requirements - for which purpose is the model needed?).
3. Implementation of the model, Section 3.3
   (which detail is necessary for the modeling task, based on which data?).
4. Calibration and validation of the model, Section 3.4
   (determination of the parameters of the model based on available data sources or measurement data).
5. Usage of the model.

The different steps in the modeling process are often performed by different persons, either in the same company or also across different companies (for example supplier and OEM). Thus, in order to have a traceable model, all relevant information has to be managed during the whole modeling process. Therefore, a source-code management (Section 4.1) and a version management (Section 4.2) is necessary and the utilized resources need to be documented (Section 4.3)

One simple approach is to use a template with the above-mentioned items (for example as Word or as Markdown document) and to store the filled-out template in the model, for example as file `model_history.md` in the

[1] https://fmi-standard.org/
[2] https://www.upsim-project.eu/

DOI 10.3384/ecp21181359

root directory of the relevant Modelica package. For every item a short summary must be included in the template together with a description where the full details can be found. This might be e.g., a report in the *Resources* directory of the package (cf. Section 4.3), a publication, an internal report of the organization, or repositories or file servers where simulation results or measurement data are stored.

In the future, it would be also very helpful if the log-comments of a version management system are automatically extracted and attached to the model documentation, in order to get easy access to this, usually, very valuable information. In open-source projects the release information of a software contains often a one-line comment and a link to every issue and/or pull/merge request for this release. The same process could be also done for models (Section 4.2).

Another approach to track changes of artefacts during the modeling process is presented in (König et al. 2020). The changes can be tracked more or less automatically by collecting standardized information, which is sent by the different tools involved in the modeling process, in a database using server communication. Unfortunately, the number of tools currently supporting this traceability approach is currently small, but is planed to be extended.

The focus of this paper is the development of a model until a state “ready to use” is reached. The usage of the model itself, including the maintenance of the model throughout its whole life-cycle, is, in contrast, not considered in this paper.

3 Model development

3.1 Requirements

As described in Section 2, the modeling process starts with the definition of the requirements to be fulfilled by the final model. Requirements are typically developed and defined textually in a document-based development process using natural-language that might be following some rules, such as the Easy Approach to Requirements Syntax (EARS), see (Alistair and Wilkinson 2019). As a typical example, see the requirements MIL-STD-704F for electrical systems in US military aircraft (Department of Defense 2016). Definitions of requirements with natural language might be supported with appropriate tools, such as DOORS [3] from IBM or Reqtify[4] from Dassault Systèmes, to get support for *collaboration, traceability* and *coverage analysis*.

There are several proposals and attempts to define and check requirements more formally, such as (Schamai 2013) where it is proposed to generate Modelica code for this purpose. At Electricité de France (EDF), the special language FORM-L – FOrmal Requirements Modelling Language, (Thuy 2014), was developed that *formally* defines requirements in a language close to the textual notation used by system designers. The FORM-L language is centered around the four basic questions: *What*, *Where*, *When*, *How Well*. The example from (Bouskela and Jardin 2018) maps the natural language requirement

> R1: While the system is in operation, the pump must not be started more than twice.

into the following FORM-L definition:

> **requirement** R1 **is**
> **for all** pump **in** system.pumps
> **during** system.inOperation
> **check**
> **count** (pump.isStarted **becomes** true) <= 2

FORM-L uses two and three-valued logic to define the logical parts of requirements. The reason to use three-valued logic is that in certain situations it is not possible to state whether a property is *satisfied* (= true) or *violated* (= false) and, therefore, a third value type *undefined* is introduced.

In order to make the FORM-L language directly accessible for the Modelica community, the open source Modelica library *Modelica_Requirements*[5] was developed (Otter et al. 2015). The library has about 200 model and block components and about 50 functions. It allows to define requirements with drag & drop and to “bind” these definitions to Modelica models, so the requirements are always checked when the models are simulated. It is then reported, whether requirements are satisfied, violated, or not tested. Due to the needs of the *Modelica_Requirements* library, some additional functions have been introduced in the Modelica Standard Library (MSL)[6], in particular the functions of sub-package *Modelica.Math.FastFourierTransform*.

In (Bouskela and Jardin 2018), the new Extended Temporal Language (ETL) is described for the simulation of the temporal aspects of FORM-L. ETL introduces a four-valued logic by the additional value *undecided* (meaning that the “decision making” is in progress), in contrast to *undefined* (meaning that the “condition” is not applicable). There is also the Modelica package *ReqSysPro* under development at EDF to practically use ETL within a Modelica model. For more details about the usage of FORM-L in the Modelica community, see also (Bouskela, Falcone, et al. 2021).

The abovementioned approaches define requirements formally and perform simulations on Modelica models to automatically retrieve answers whether requirements are satisfied, violated, or whether no answer can be given. In order to make that possible, corresponding scenarios must be defined (see Section 3.2). Thus, particular simulation runs must be selected to perform these tests. It might not be obvious to determine suitable scenarios. Instead, the

[3] https://www.ibm.com/products/requirements-management

[4] https://www.3ds.com/products-services/catia/products/reqtify/

[5] https://github.com/modelica-3rdparty/Modelica_Requirements

[6] https://github.com/modelica/ModelicaStandardLibrary

central question is to figure out the scenarios and environmental conditions for which the requirements are not fulfilled, for example to figure out a (valid) load condition or an operating point, where a controlled system is unstable.

A standard approach is to use *Monte Carlo Simulations*, generating parameter and initial values randomly according to given distributions and perform simulations for every randomly selected value set. Most Modelica tools support Monte Carlo Simulation. There are also dedicated tools, such as Persalys[7]. The drawback for typical industrial usage scenarios is, that the number of parameters and initial values is so large, that it is easily possible that operating points are not found where requirements are violated. An alternative is to utilize more intelligent search processes:

- Since 1997, *worst-case optimization* is routinely used at DLR's Institute of System Dynamics and Control to tackle such problems. Hereby, multi-criteria optimization problems are defined so that systems behave as worse as possible. For more details, see e.g. (Bals, Fichter, and Surauer 1997; Joos 2015; Labusch et al. 2014).
- (Corso et al. 2020) provides a survey of optimization algorithms for black-box safety validation: "[..] finding disturbances to the system that cause it to fail (falsification), finding the most-likely failure, and estimating the probability that the system fails".
- TestWeaver from Synopsis[8] (Tatar and Mauss 2014) automatically generates stimuli for co-simulations of virtual ECUs, plant models and requirement watchers. The main generation strategy is the Coverage-Driven Generation which combines random, combinatorial and optimization strategies with the goal to increase (discrete) coverage measures and to find worst-cases for (continuous) quality measures.

3.2 Simulation scenarios

The next step in the modeling process is the definition of the modeling task. This implies a definition of the simulation scenarios, the model should be used for. Human-readable Modelica code is very well suited for compact storage of defined scenarios. But, there are many ways to handle static and dynamic boundary conditions of a Modelica model and to define test cases.

3.2.1 Parameters

When preparing a model for a specific simulation task, the source of parameter values should be documented: Who changed the default value and why? At the instance level, changing parameters means introducing *modifiers*. Modelica modifiers do not provide the ability to comment modifications. A good practice in larger Modelica projects is to define at which hierarchical level the parametrization should happen. Therefore, if there is a defined level, the source of parameters can be documented on Modelica info layer: component data sheets, measurements, educated guess, optimization results, etc. Another approach is to store parameters with this documentation in replaceable hierarchical records in a separate Modelica library or sub-library on a different file as the model, in order that changes to parameters are directly/easily visible in the version control system.

The parameter handling of Modelica should be improved to better support the formal definition of credible models. In principle, extensions could be introduced via *Custom Annotations* (Zimmer, Otter, and Elmqvist 2014), but large scale usage is currently not user-friendly. Furthermore, a standardized solution is needed, supported by all the Modelica tools. In particular, the following features would be useful:

- Defining the domain of validity of the model, preferably with a language element and not just in the documentation.
- Define parameter tolerances and/or uncertain distributions (such as normal or uniform distribution).
- Introduce an orthogonal concept to parameter propagation by mapping parameter values, their tolerances and distributions into a model, so that model structure/equations and model data can be much easier separated. This could be done by the *merge* concept proposed in (H. Elmqvist et al. 2021).

3.2.2 Boundary conditions

A specific system model can be used in different scenarios: a) simulation of stationary load points, b) simulation with dynamic boundary conditions, or c) with external boundary conditions (e.g. co-simulation or hardware-in-the-loop (HIL)).

The dynamic boundary conditions can be important for the correct initialization of a model. One example would be to initialize a system based on the correct environment temperature and pressure. From our past experience, it is hard to initialize based on external dynamic inputs, e.g. coming from table data or via co-simulation. This is, because in Modelica, the start values of input variables can not be directly used as initialization parameters (parameters having lower variability as inputs). To overcome this problem, users have to write additional initial equations in order to assign input values to initialization parameters, see example code in Listing 1.

If the equations allow initalization of dynamic states via connectors, graphical solutions like `Modelica.Mechanics.Rotational.Components.InitializeFlange`[9] are possible.

[7] https://persalys.fr

[8] https://www.synopsys.com/verification/virtual-prototyping/virtual-ecu/testweaver.html

[9] https://doc.modelica.org/Modelica%204.0.0/Resources/-helpDymola/Modelica_Mechanics_Rotational_Components.html

Listing 1. Initalization of states from output of CombiTimeTable

```
...
  parameter SI.Temperature T_initial(fixed=false) "Init, to be set from table output";

  Modelica.Thermal.HeatTransfer.Components.HeatCapacitor heatCapacitor(
    C=100, T(start=T_initial)) "Component to be initalized";

  Modelica.Blocks.Sources.CombiTimeTable combiTimeTable_Tamb
    "Boundary condition on table file";

initial equation
  T_initial = combiTimeTable_Tamb.y[1] "Get first time point of table data";
...
```

The open source Modelica library `ExternData`[10] from Thomas Beutlich and further contributors is very helpful in this respect, because it allows to read data from CSV, INI, JSON, MATLAB MAT (v4,v6,v7,v7.3), SSV (System Structure Parameter Values v1.0)[11], TIR (Tire properties), Excel XLS and XLSX, and XML files at the start of a simulation *without newly compiling the model.* A Claytex TechBlog[12] gives a good description how to use this library. Data from INI files can be read in a similar way with the open source Modelica library `DeviceDrivers`[13].

3.2.3 Test case definition

For the definition of *test cases*, usually a scripting language is needed to describe the wide variety of occurring situations. There is neither a standardized nor an accepted scripting language available in the Modelica community. Depending on user's preferences and the support of scripting languages in the used Modelica tool, one of the following scripting languages is typically used: Dymola functions or script files (mos-files), Microsoft Excel (called Excel in the following text) via Excel-plugins such as XRG Score[14] or TLK Simulator for Excel[15], Maple, Mathematica, Matlab, Python. Industrial users have often a strong preference for Excel or Python.

A new, interesting possibility is proposed in (Buse and Bellmann 2021) where one or more instances of a Lua interpreter[16] can be attached to a Modelica model via the Modelica external object interface. Hereby complex scenarios can be defined in a combination of Lua- and Modelica-code. The Lua interpreter itself is a small DLL (Dynamic Link Library) that is included in a Modelica library and therefore does not require any download or installation, contrary to other scripting environments that are used with Modelica tools.

The following examples demonstrate how Dymola's scripting can be utilized for the definition of test cases.

Example 1: Automotive driving maneuvers There exists a range of well defined standard scenarios to identify and verify the particular behavior of a vehicle. They are typically referred to as open loop or closed loop driving maneuvers. The former comprises for example ISO 4138 steady-state cornering, the latter ISO 3888 double lane change. The example provided here uses Dymola built-in functions.

Considering a vehicle's architecture as defined in the *VehicleInterfaces* library[17](Dempsey et al. 2006), see Figure 1, the particular driving maneuver conditions can be often controlled within the driver model only. Making this element replaceable enables to adapt the desired driving conditions from outside, e.g. when simulating the model. Utilizing this feature, there can even be implemented a function to simulate various driving maneuvers in one turn.

In the predefined vehicle's architecture, called `ConventionalVehicle` in Listing 2, a template of a vehicle model is put together with its environment. Therefore, a particular user's vehicle model shall be redeclared here and the architecture shall be checked against possible compiling errors in a pre-processing step. Then, the function in Listing 2 can be executed with particular settings for each of the desired maneuvers. The two exemplary maneuvers, mentioned above, are given in Listing 2.

Listing 2. Run driving maneuvers

```
function runManouvers
  input String modelSim =
    "VehicleLibrary.ConventionalVehicle"
    "Model to be simulated"
    annotation (
      Dialog(
        __Dymola_translatedModel(
          translate=false)));
  input String priorRedeclarations = ""
    "User defined prior modifications, e.g.
    'redeclare class block(par1=...),'";
  input String resultFile = "myVehicle"
```

[10] https://github.com/modelica-3rdparty/ExternData

[11] https://ssp-standard.org/

[12] https://www.claytex.com/tech-blog/using-external-data-in-your-dymola-model/

[13] https://github.com/modelica-3rdparty/Modelica_DeviceDrivers

[14] https://www.xrg-simulation.de/de/produkte/applications/score

[15] https://www.tlk-thermo.com/index.php/en/simulator-suite

[16] https://www.lua.org/

[17] https://github.com/modelica/VehicleInterfaces

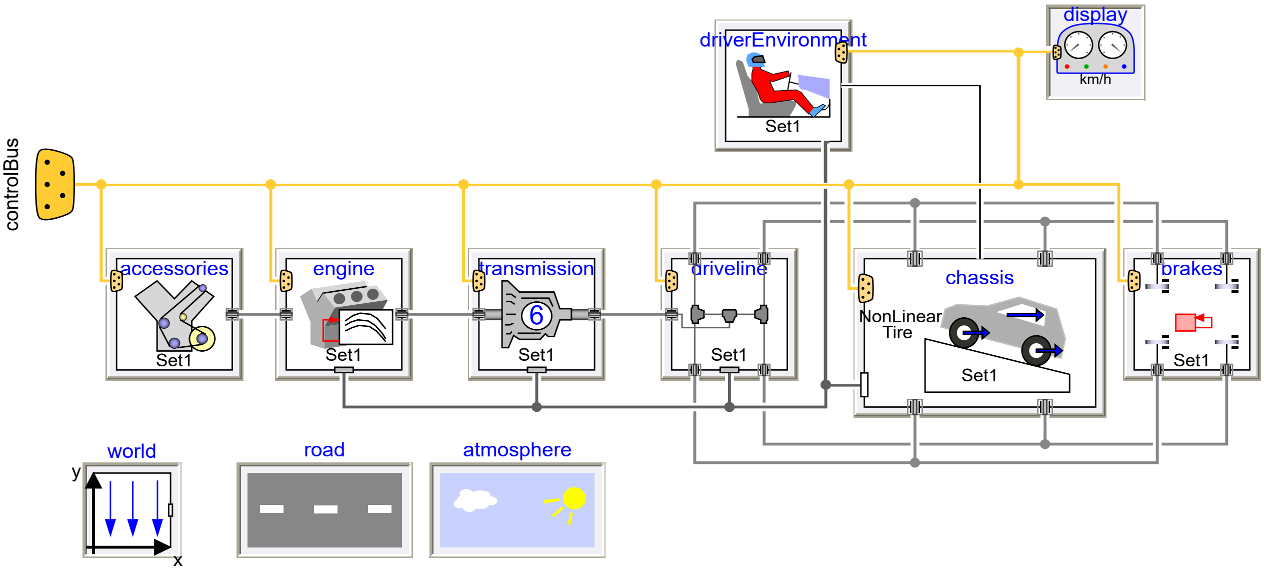


Figure 1. Common vehicle's architecture as defined in the VehicleInterfaces library and inherited in the PowerTrain library.

```
    "Result file(s)";
  input Boolean cornering40 = false
    "ISO 4138 Cornering: R = 40 m";
  input Boolean laneChange80 = false
    "ISO 3888-1 Double lane change: v_x =
       80 km/h";
  ...
protected
  String problem;
  String modifier;

algorithm
  if cornering40 then
    modifier :=
      "VehicleLibrary.Drivers." +
      "SteadyCornering" +
      " driverEnvironment(" +
      "vInit=2,Rcurve=40)";
    problem :=
      modelSim + "(" + priorRedeclarations
        + "redeclare " + modifier + ")";
    ok := simulateModel(
      problem = problem,
      startTime = 0.0,
      stopTime = 250,
      resultFile = resultFile
        + "_StationaryCircle40m");
  end if;

  if laneChange80 then
    modifier :=
      "VehicleLibrary.Drivers.LaneChange" +
      " driverEnvironment(" +
      "vInit=22.22,variant=ISO3888_1)";
    problem :=
      modelSim + "(" + priorRedeclarations
       + "redeclare " + modifier + ")";
    ok := simulateModel(
      problem = problem,
      startTime = 0.0,
      stopTime = 15,
      resultFile = resultFile
        + "_DoubleLaneChange80kmh");
  end if;
  ...
end runManouvers
```

The core of the function are single simulation calls of Dymola's `simulateModel` function for each of the maneuvers, carried out for the predefined `ConventionalVehicle` model. Particular maneuver conditions are given by the predefined attribute `modifier`. Furthermore, maneuver specific simulation conditions are set, such as the simulation stop time. Moreover, each driver model contains an option to stop the simulation once maneuver-specific conditions are reached – for example the maximum lateral acceleration for the steady-state cornering.

Besides fixed maneuver-specific attributes, the attribute `priorRedeclarations` additionally enables to modify the remaining blocks of the `ConventionalVehicle`. Thus, for example variants of power train, environment or controller can also be defined.

Example 2: Refrigeration cycle When simulating refrigerant cycles, there are usually subsequent tasks performed interactively or via scripting: The first step might be a parameter variation to define a suitable refrigerant charge. Then, stationary results can be verified, e.g. by checking the pressure-enthalpy-diagram. After this verification step, larger simulations studies on control points and environmental parameters, like air temperature, moisture and speed, can be defined.

3.3 Model complexity

After the definition of the simulation scenarios, the complexity of the models should be considered. The decision about complexity is based on the scenarios, the model is implemented for.

3.3.1 Adequate modeling detail

Models describing every effect of a (technical) system, don't exist. A model is always created for a specified set of scenarios. Utilizing the scenario-driven model for another scenarios can either lead to inappropriate quality of results, because the desired effects are not modeled, or it needs an unnecessary parametrization effort and a lot of computational resources, because it is too complex for the expected outcome. An example of a mismatching vehicle model is shown in Figure 2. Assuming a line-change maneuver as a scenario, both submodels are inappropriate. With the single-track model the roll angle of the vehicle can not be evaluated and the FE-model, calculating the deformation of the rim, is far too detailed for this scenario. Instead it can be assumed to be rigid .

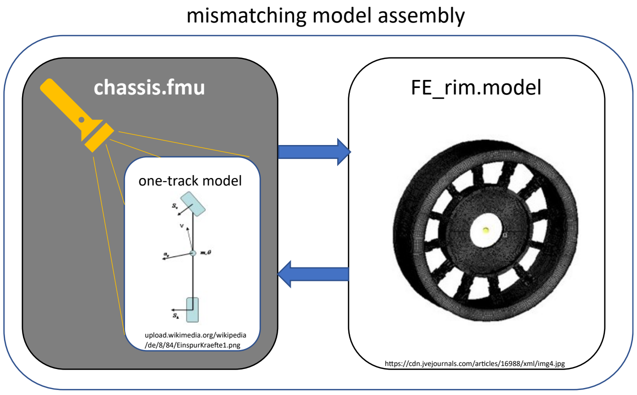

mismatching model assembly
chassis.fmu
one-track model
upload.wikimedia.org/wikipedia
/de/8/84/EinspurKraefte1.png
FE_rim.model
https://cdn.jvejournals.com/articles/16988/xml/img4.jpg

Figure 2. Mismatching model assembly. A simple single-track model of a vehicle (left) versus a complex finite element model of a wheel's rim (right, from (Wei et al. 2016)).

The accuracy of the worst submodel in an assembly (e.g. the less detailed) generally determines the total accuracy. So it's reasonable to assemble models of similar complexity, depending on the aim of the use case. In the example either a rigid rim or a more complex chassis-model should be used. The adequate complexity of a model also depends on the desired outcome of the whole system. If the model has only a small influence on the outcome, it can be modeled less complex.

A model developer should keep some questions in mind while creating a scenario-driven model:

- What is the desired output of the model?
- Which accuracy does the model need?
- Which accuracy has the data used for calibration and validation?
- Based on which data can the model be validated?

If a model is encapsulated as a FMU, its complexity is not known in general if the source-code is not published. In the above-mentioned example, this might be the reason why the two models of largely different complexity are even assembled. The complexity can be estimated by simulating the FMU and regarding the output (e.g. the translation statistics of the tool). This additional effort could be avoided if a "scale" for the model complexity could be stored in the FMU.

3.3.2 Model detail levels

The detail level of a model can be roughly characterized by the following classification (adapted from (Kuhn, Otter, and Raulin 2008), which in turn is based on a classification sometimes used in aerospace industry):

- Level 1: Architectural level
  Steady-state power consumption.
  Models are described by *algebraic equations* based for example on the energy balance between ports (without dynamic response). Typical use: Rough system design with power budgets.

- Level 2: Functional level
  Steady-state power consumption and mean-value transient behavior (e.g. inrush current or consumption dynamics with regard to input voltage transients).
  Models are described by *differential-algebraic equations* (without switching elements). Typical use: Stability studies, controller design.

- Level 3: Behavioral level
  Detailed description of transient behavior (e.g. switching and high frequency injection behavior).
  Models are described by *hybrid differential-algebraic equations* with events and switching elements. Typical use: Network power quality studies, verification of controllers.

- Level 4: Distributed level
  Very detailed description of spatially distributed, transient behavior (e.g. magnetic field in electrical motor, stress field in a structure, flow around an airfoil).
  Models are described by *partial differential equations*, which are solved with FEM (Finite Element Method), FVM (Finite Volume Method), CFD (Computational Fluid Dynamics), or DEM (Discrete Element Method). Typical use: Detailed vibration investigations, design of structures or of the windings of electrical motors.

In Figure 3, the simulation of a DC/DC buck converter is shown for model levels 1, 2 and 3. In (Kuhn, Otter, and Raulin 2008), various alternatives are discussed how to implement multi-level models in Modelica.

3.3.3 Selecting the model detail

A typical modeling process starts with a simple model for one use case and then the complexity and number of use cases is increased step by step. In the end the model contains various physical effects and cover many use cases, but for the initially anticipated simple use case it's way too complex.

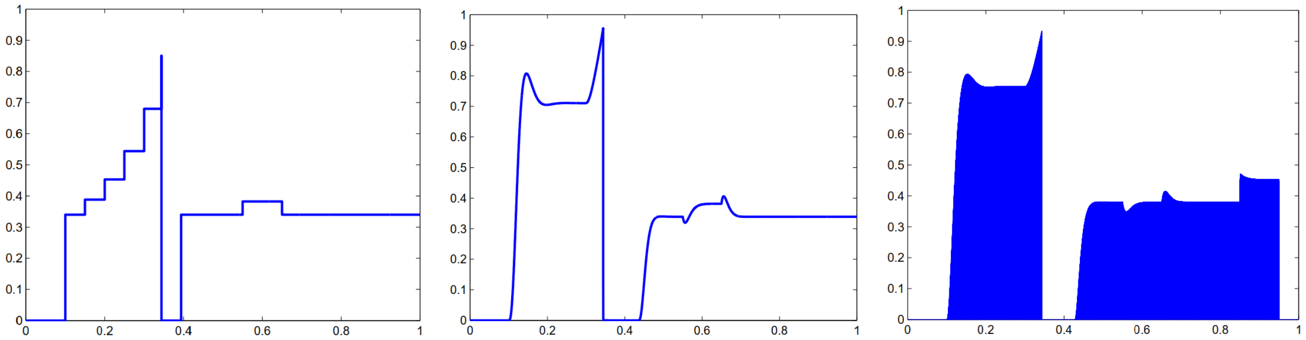


Figure 3. Exemplary simulation results of architectural (left), functional (middle) and behavioral (right) model of a DC/DC buck converter (Kuhn, Otter, and Raulin 2008). The right image contains high frequency switching leading visually to a "filled area".

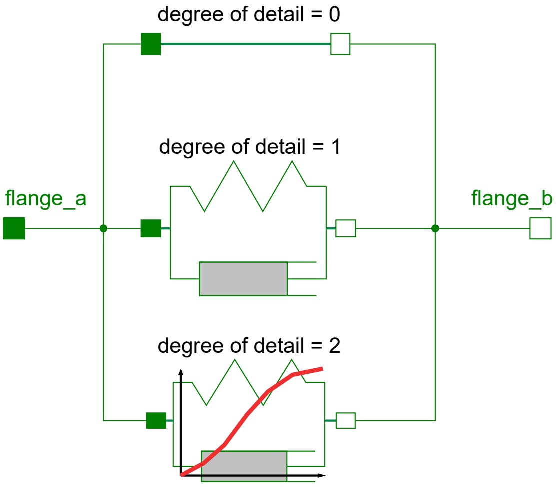


Figure 4. A Modelica model containing three conditional components reflecting different level of model's detail.

A good practice to avoid such over-engineering is to store development stages of different complexity and then choose a suitable complexity for a specific use case. The Modelica language provides the following possibilities to switch between different modeling levels:

- Partial models.
- Conditional components.
- Replaceable components.
- If-clauses.

This modeling practice also simplifies the documentation, because it's obvious to describe the differences between the levels of complexity, including the effects added to the model step by step. On the contrary, the necessary assumptions and simplifications can be described, to use one of the less detailed models.

In Figure 4, a simple example for conditional components is shown. Three different degrees of details are modeled: a rigid connection (architectural-level, cf. Section 3.3.2), a linear (functional-level) and a nonlinear (behavioral-level) spring-damper behavior between flange a and b. In this example, the particular degree of detail – "stiff", "linear" or "nonlinear" – can be selected by changing the value of the parameter `degree_of_detail`, as shown in Listing 3.

Listing 3. Conditional components

```
model ConditionalComponents
  ...
  parameter Integer degree_of_detail=0 "
      Parameter to switch between models
      with different degree of detail";

  Components.Detail_0 detail_0 if
      degree_of_detail == 0;
  Components.Detail_1 detail_1 if
      degree_of_detail == 1;
  Components.Detail_2 detail_2 if
      degree_of_detail == 2;

equation
  if degree_of_detail==0 then
    connect(flange_a, detail_0.flange_a);
    connect(detail_0.flange_b, flange_b);
  elseif degree_of_detail==1 then
    connect(flange_a, detail_1.flange_a);
    connect(detail_1.flange_b, flange_b);
  else
    ...
  end if;
  ...
end ConditionalComponents;
```

The selection of an appropriate modeling stage depends on the scenarios of the whole system. In Listing 3, the rigid connection can be used if the component, represented by this model, is very stiff in relation to other components in the system. In contrast, the linear spring behavior is not adequate, if the linear elastic range can be exceeded due to large forces appearing in the system.

Similarly, to the connect-statements inside Listing 3, if-clauses can be used to switch between different sets of equations, with the disadvantage, that each branch must have the same number of equations in Modelica. This may lead to many dummy definitions. Another option in Modelica are replaceable models. Unfortunately, they are only beneficial if the interfaces (inputs, outputs, parameters) of the different modeling stages are similar, because they have to be adapted each time the replaceable model is changed.

A disadvantage of storing different levels of detail is the difficult testing strategy and higher effort for model adaptations. Functionality tests and model updates must be performed for each level. Therefore, it is advisable to store only major development steps of different complexity.

Typically, the level of detail should not only be switchable for a system model, but also for its sub-components. Technically, such an approach cannot be implemented fully satisfactory in the current Modelica language, because a feature to replace sub-components based on Boolean expressions is missing.

If an FMU should be built, the parametrized switch between different modeling stages (which is a structural parameter) cannot be transferred, because the number of state variables cannot be changed after the translation of the model. An FMU can only contain one single degree of complexity. It would be useful to support different levels of modeling details in a future Functional Mock-Up Interface (FMI) standard.

3.3.4 Physical versus data driven models

An important indicator for the credibility and the range of validity of a model is the modeling basis. Models can be based on physical equations or on data from measurements, expert guesses or other sources.

Physical equations are usually public knowledge with well defined assumptions. They usually cover a wide range of physical applications and can be extrapolated without immediately leaving their range of validity. Unfortunately, a proper parametrization is often difficult, because finding accurate parameter values (e.g. the friction coefficient between two bodies in contact) can be costly.

Data driven models are using for example measurement data. These data contain every single influence (e.g. the environment temperature) on the system during the measurement – even unknown ones. Consequently, the model is only valid for exactly these circumstances and, moreover, its extrapolation is limited. To put measurement data into a Modelica model so called Combi-Tables can be used. In this case the option "extrapolation triggers an error" of the Combi-Tables should be selected.

Characteristic maps based on measurement data also often induce a higher numerical effort for Modelica tools. This is because interpolation between the data points is necessary, non resolvable non-linearities can appear, and the variable described with the characteristic map can not be selected as a state-variable.

A model based on measurement data must contain all relevant information about the measurement (such as measured range, measurement methods and tools, measurement precision, ...) best in a formal way, and not just in the documentation.

Other kinds of data driven models can use mathematical approximations such as neural networks, response surfaces or optimization functions. Mathematical approximations are either used as a simplification of a physical model to reduce computation time or as an attempt to generalize measurement data by learning from the output of multiple measurements. There are several publications on this topic, also from the Modelica point of view (e.g. (Bruder and Mikelsons 2020) and (Tundis et al. 2017)).

3.4 Model Calibration

Once a model is implemented, the modeling process goes on with the calibration, validation and verification, to ensure that the model appropriately achieves the aims it was made for. Since the scope of model calibration, validation and verification is very large, we can only give here a short overview in the context of a typical use case for a multi-physical Modelica model. For a broader overview on the topic, a recent paper (Riedmaier et al. 2021) goes into more details and gives many additional references.

Multi-physical Modelica models are typically used either to represent a real physical system or to reproduce the behavior of such a system as a part of a model-based feed-forward or feed-back control system. Especially the direct generation of inverse models from Modelica models is a powerful feature which can also be used during the calibration process, see (Reiner 2011) or (Mesa-Moles et al. 2019).

3.4.1 Goal definition

The aim of the calibration process is to parameterize models with the help of measurements with regard to defined goals and criteria. For models, this generally means that they should map the behavior of the real-world system as precisely as possible. Figure 5 shows an overview of the model calibration process.

The calibration of the parameters of these models is important in order to achieve a good representation of the real system or to have a good controller performance, in the case of model-based control.

For many physical systems the knowledge of the individual involved parameters can be quite different. For example, for the model of a robot, the mechanical parameters such as link lengths or masses could be known very precisely from data-sheets or CAD data, whereas the friction or damping in connecting joints can be highly unknown. Well known parameters should, thus, be included in the models directly, in order to reduce the number of unknown parameters for the calibration process. Additionally, the source of these parameters should be documented.

The aim of the calibration for model-based controllers is an optimal control performance, as well as sufficient disturbance suppression and vibration damping (in the case of feed-back control). For Modelica models used as part of a control system, this often means that the model is calibrated directly on a HIL (Hardware-in-the-Loop) setup (Reiner 2011).

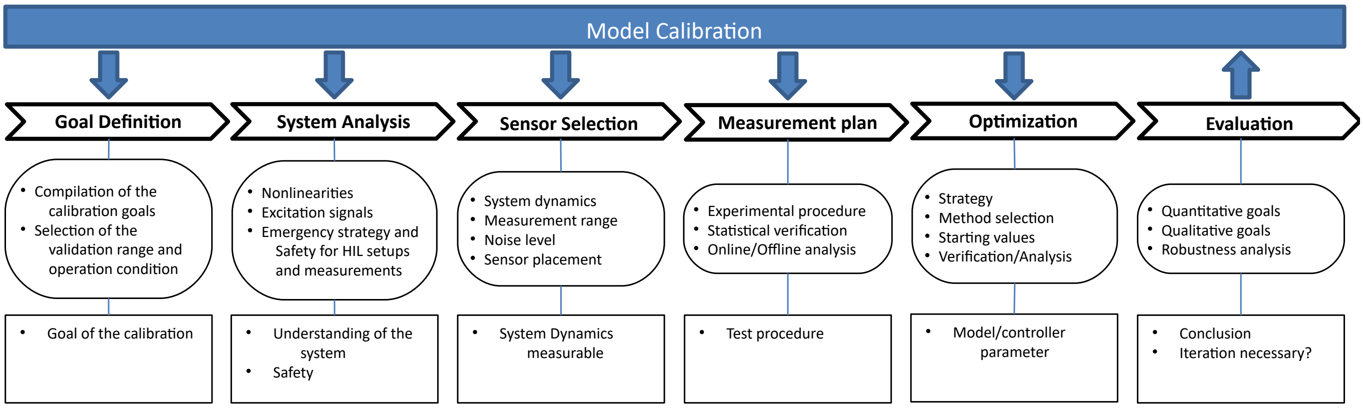


Figure 5. Overview for the Modelica model calibration process using measurement data.

3.4.2 System Analysis

There are numerous approaches for the identification process in the literature. In particular, however, a distinction must be made between methods for verification and identification in the frequency domain and in the time domain. Methods in the frequency domain are useful for systems that have (approximately) linear behavior, or for individual operating points of a system for which this assumption applies. Methods in the time domain are also suitable for non-linear systems, but are usually associated with greater effort (e.g. with regard to required computing time and experiment effort / duration).

Multi-physics Modelica models are usually used for modeling of complex non-linear systems. However, also the generation of linear systems from Modelica models is possible using numerical linearization, which is supported by many Modelica tools. Nonetheless, the focus is on non-linear models in the following.

In a second step of the calibration the physical system has to be analyzed after the aim of the calibration has been be defined.

Normally, models of a physical system do not contain every detail or the entire operating range of the system, see Section 3.3.1. It must, therefore, be investigated to what extent an undesired or non-modeled behavior can be isolated. For the verification of model-based controllers on (HIL) test benches, further considerations must be made, such as taking precautions to ensure safe operation of the test bench in case of a controller failure (e.g. unstable behavior or violation of manipulated variable restrictions). In addition, suitable emergency strategies (e.g. emergency stop switch, mechanical emergency braking) must be implemented. If the controlled system is unstable without a controller, a robust parallel controller can also be helpful, which can be activated in the event of a fault and bypasses the controller to be verified.

3.4.3 Sensor selection

Suitable sensors have to be selected based on the calibration objectives. It is important to consider the dynamics and measuring ranges of the sensors used. If no single sensor can capture the entire relevant dynamics of the system, it must be examined whether a suitable result can be achieved by merging several measurements and / or sensors. Sensors generally have measurement noise and offsets, that must be considered during measurements and appropriately compensated / calibrated. This can also be done during a pre-processing step. After an analysis of the system, the sensor placement must be selected in such a way that the dynamics of the system is reproduced as clear as possible. This is especially important for elastic mechanical systems (e.g. eigenmode shapes). In case of doubt, different placements should be examined (in the case of elastic systems, the measurements can be influenced by vibration modes, for example).

3.4.4 Measurement plan

After suitable sensor selection and placement, a measurement plan should be documented, for which the statistical nature of measurements must also be considered. Critical measurements (e.g. measurements of parameters with a large impact on the system dynamics) must always be carried out several times and, if there is a broader spread of the measurement results, they must also be processed appropriately. Particularly in the case of large deviations, the sensor selection and placement must be critically examined again.

For the verification and tuning of controller parameters directly on the test bench, online methods are available, in which the evaluation takes place directly after or already during the measurements on the test bench. HIL setups are suitable for this, in which the Modelica-based model controller parameters can be changed with minimal effort.

Alternatively, however, offline methods can also be used in this case by first identifying suitable Modelica models of the controlled system in order to be able to design appropriate controllers with the help of simulations. For the identification of Modelica models and model-based observer systems of the physical system, an “offline” method can always be used, because the change of the Modelica models’ parameters does not change the measurement data, which is, on the contrary, the case, if the model is part of a feed-back system. This enables more elaborate methods to be used. Since Modelica models typically also do not represent the physical system up to very high frequency ranges, noisy measurement data should be low-pass filtered before the calibration process if possible using forward-backward filtering to avoid a phase shift in the data.

3.4.5 Optimization

Suitable identification strategies for Modelica models are optimization-based methods. Therefore, mathematical criteria are to be defined which can then be minimized using a suitable optimization algorithm by varying the parameters of the model. Appropriate start parameters have to be selected for this purpose. An important difference to classical optimization, however, is the statistical nature of the measurement results, which must be appropriately considered for the criteria specification. In the case of HIL optimization, mainly only optimization methods with a small number of steps (function evaluations) are to be used, since HIL experiments are usually significantly more time-consuming compared to pure numerical evaluations. This normally means that local optimization methods are to be preferred (e.g. gradient based methods or surrogate optimization techniques).

For offline methods, nearly all optimization algorithms can be used. However, because of the noise in the data, usually those algorithms converge better, which are gradient-free or robust against noise. For Modelica models, a wide range of optimization tools are available. There are methods for optimizing the Modelica model directly within Dymola using e.g. the DLR *Optimization* library (Pfeiffer 2012), as well as many other external tools. The latter use the Modelica model directly as an executable or exported as an FMU within a chosen environment, such as Python or Matlab, since the parameters of the Modelica models can still be changed, even after the compilation and export, see e.g. (Leimeister 2019).

3.4.6 Evaluation

After the Modelica model of the system has been verified the obtained results must be assessed quantitatively and qualitatively with regard to the selected objectives. To ensure robustness and to avoid over-fitting, additional measurements should be used for this step, which were not used within the optimization process of the parameters. If not all goals could be met during the calibration process, the process must be carried out again iteratively after an analysis of the results and, if necessary, a new modified model or controller structure must be used.

As a final result, the obtained set of model parameters should be documented, together with the model and the original measurement data, as well as a detailed description of the overall process. For Modelica models, such a documentation can be done directly within the model (see also sec. 3.2.1).

4 Model management

The modeling process described in Section 3 is a complex procedure with many steps and iterations. During the modeling process as well as afterwards, the different modeling stages should be traced. Therefore, a version and revision management is needed, as well as an archiving of measurement data, simulation results, etc.

4.1 Source code management using *git*

Modelica models and packages are stored in ASCII-files. It is therefore obvious to store these files in a source code management system, such as *GitHub*[18] or *GitLab*[19], which provide a lot of additional functionality compared to pure *git*[20]. Model developers can then profit from issue tracking, merge/pull requests, release handling, etc. One disadvantage of these widespread source code management systems is that access rights can be only defined for a whole repository. In industrial projects, it is typically not desired that everyone has access to the whole information and it is then necessary to split the information over several repositories, for example:

- A repository contains the source code of the Modelica model or the Modelica package. Only model developers have access to this repository.
- A repository contains the released versions (possibly with encrypted Modelica packages), as well as an issue tracker. Users of the model library have access to this repository.
- A repository contains administrative information, such as contracts, clearance, license information. Managers have access to this repository.

Data to parametrize the models is often *not* stored in a source code management system but is extracted from database systems. Simulation results and measurement data is often stored in binary format and, therefore, a source code management system is not well suited. Instead, this data is typically stored at different locations on a pure file system without version management. Altogether this means that the information about a model might be spread across several locations. Consequently, it is advisable to maintain a document with information about the different storage locations. Regarding the model credibility, there is room for improvement.

Due to its textual nature, any change in the Modelica model can be traced, read and understood by humans. But, keeping minimal textual differences is a hard job for Modelica tools, especially when users mix graphical and textual modeling. As the formatting of Modelica code is not specified, the problem even increases if different Modelica tools are used within one project. To our experience, it

[18] https://github.com/

[19] https://about.gitlab.com/

[20] https://git-scm.com/

is a significant early step to establish implementation rules for a project collaboration in order to keep model changes traceable. A public starting point for these modeling conventions is `Modelica.UsersGuide.Conventions`[21]. Another best practice concerning Modelica code management is to seperate git commits for larger graphical and documentation changes and for actual changes to the model (like introducing new components or changing or adding equations). Furthermore, all auto-formatting in tools should be switched off.

4.2 Version management

While source-code management of Modelica models works fine within one organization, most of the time this important information is lost when delivering models across organizational boarders, e.g. from supplier to OEM. Modelica has multiple ways of storing the current revision information in annotations e.g. the `revisions` and/or `revisionID`. See an example of MSL 4.0.0 in Listing 4. One important limitation is that `revisionId` and `dateModified` are usually stored only on the top level package, so it cannot be used to see when an actual class has been changed. For specific classes, one has to rely on the non-formalized `revisions` annotation. Conventions for how to structure and update the `revisions` annotation are library specific, so far. The `revisionId` is not automatically handled by most Modelica tools and it is typically lost when generating an FMU based on Modelica code.

While Modelica tools can give support on versioning and releases, a library release still implies coordination between multiple team members. One public resource for release workflows is the *Modelica Standard Library Developers Wiki*[22]

4.3 Model resources

By convention, non-Modelica language information, such as manufacturer data sheets, data-files for CombiTables, images, is stored in folder *Resources* which is located at the top level directory of a library. It might be helpful to distinguish – also in the structure of the Resources folder – between *model-data* (parametrization data, e.g. for CombiTables) and *documentation-data* (e.g. example results or information used for modeling). Model-data are necessary for the functionality of the model, while documentation-data is “only” nice to have, but can be extremely helpful for the comprehension of the model. There is a smooth transition between model-data and documentation-data. For example, icon-graphics is irrelevant for the functionality of a library, but improves the user’s handling significantly by giving a quick graphical impression of the component.

As already mentioned in Section 4.1, larger measurement databases are usually not stored within the Modelica library. But, in order to understand the quality of calibrated models, we propose to store a minimal set of measurement data, actually used for parameter optimization within the Resources folder as documentation-data.

While the file structure of Modelica code is automatically updated by a Modelica tool on the file system, the central resources folder has to be organized manually. This is error-prone and, depending on the chosen structure, hard to update when e.g. re-structuring the Modelica package. External links to larger measurement data sources is not handled by the current Modelica package concept and local resolving of the `loadResource()`[23] function.

In order to be able to extract working system models out of a larger library, Modelica tools are typically able to store a so-called *total model*, including all required resources. The storage of total models is not standardized in Modelica, yet. The `loadResource()` function in combination with a Uniform Resource Identifier (URI) helps to avoid path issues and allows a Modelica tool to analyze which resources are needed. As models can access files in multiple ways, it still remains a tedious task to check e.g. for missing resources or too much files to be copied.

4.4 Archiving of simulation results

Simulation results are usually stored in a file, in binary or ASCII format. This includes reference data as input for a model, as well as reference results that a simulation should reproduce within some tolerance. Various result file formats are used in the Modelica community, but there is no satisfactory, standardized solution.

When results need to be exchanged, such as reference results of the Modelica Standard Library, or of FMUs, they are often stored in CSV format[24] due to its widespread support in tools. Hereby, the result is seen as a table, where every column has a name (optionally with "."s to mark hierarchical structures or "[..]" to mark elements of an array) and represents a time series. The first column contains the monotonically increasing value of the independent variable (usually `Time`). A discontinuity is signaled by two identical time instants. For an example see listing 5.

Listing 5. Example of a CSV file with time event at 0.1 s

```
Time, control.w_ref, motor.w, on[3]
0.0,      0.0       ,    0.0,  0
0.1,      0.0       ,    0.0,  0
0.1,      1.0       ,    0.0,  1
0.2,      1.0       ,    0.1,  1
0.3,      1.0       ,    0.2,  1
```

The essential advantage of this format is its simplicity, but there are numerous drawbacks. Especially, it is not suited for large data sets as needed to archive simulation results.

[21] https://doc.modelica.org/Modelica%203.2.3/Resources/-helpDymola/Modelica_UsersGuide_Conventions.html

[22] https://github.com/modelica/ModelicaStandardLibrary/wiki

[23] Modelica.Utilities.Files.loadResource: https://doc.modelica.org/Modelica%204.0.0/Resources/helpDymola/-Modelica_Utilities_Files.html#Modelica.Utilities.Files.loadResource

[24] https://en.wikipedia.org/wiki/Comma-separated_values

Listing 4. Annotations relevant to versioning using the example of the Modelica Standard Library

```
package Modelica

// Sub-packages removed

annotation (
  version="4.0.0",
  versionDate="2020-06-04",
  dateModified = "2020-06-04 11:00:00Z",
  revisionId="6626538a2 2020-06-04 19:56:34 +0200",
  uses(Complex(version="4.0.0"), ModelicaServices(version="4.0.0")),
  Dymola(checkSum="469888163:3572996634"),
  conversion(
    from(
      version={"3.0", "3.0.1", "3.1", "3.2", "3.2.1", "3.2.2", "3.2.3"},
      script="modelica://Modelica/Resources/Scripts/Conversion/ConvertModelica_from_3.2.3
          _to_4.0.0.mos")),
      ...);
end Modelica;
```

For larger data sets often the *dsres* (dynamic system results) storage format is used that was developed around 1996 by Martin Otter and used by Dymola. It was later also used by OpenModelica[25]. There are several importing and exporting scripts available, especially for Matlab and for Python[26]. The dsres-format consists of a set of matrices that are either stored in MATLAB MAT v4 binary format or in a textual format. The logical view is:

1. String vector **name** contains the names of the signals. An index i of this vector characterizes the corresponding signal i.

2. String vector **description** contains a description text for the signal, typically with its unit.

3. Integer matrix **dataInfo** contains information where and how a signal is stored: A signal i is stored in a matrix j in column k with an interpolation type l and an extrapolation type m. If k is negative, column $|k|$ has to be multiplied with -1.

4. The core data is stored in **data_j** matrices where every column of a matrix contains the time series of one signal. The first column is the independent variable. Different matrices can have different time axes, that is, different number of rows. Typically, two data matrices are present: One matrix with two rows, that stores the parameters as time series with two time points, and one matrix with the time-varying signal data that corresponds to the data stored in CSV file format.

Due to the connector definition, a Modelica model has typically many variables that are identical or have opposite sign. The time series of these signals are stored in a compact way with the dsres-format, because the actual time series of variables that are related by equations $v_1 = v_2 = -v_3 = -v_4 = ...$ are stored in *one* column of a data matrix. If all variables of a Modelica model are stored in a result file, then often the size of the file is reduced by a factor of 4-5 by this technique. Furthermore, the second data matrix is stored in such a way, that the binary result file can be recovered, even if a simulation run crashes during integration.

There had been a few attempts to define a standardized time series file format based on HDF5[27], an open source file format that supports large, complex, heterogeneous data and meta information stored hierarchically in one binary file: In particular, the MTSF format (Modelica Association Time Series File Format) (Pfeiffer, Bausch-Gall, and Otter 2012) and the SDF format (Scientific Data Format)[28]. In (Pfeiffer, Bausch-Gall, and Otter 2012) it is reported that simulation results up to 200 Gbyte could be stored and retrieved in HDF5 format on file. Although, HDF5 looks attractive for scientific data sets and especially simulation result data, it has severe drawbacks, especially because it is complex and not suited for today's cloud-services. For a more detailed discussion, see (Tiller and Harman 2014).

A more modern design is the recon[29] format developed by (Tiller and Harman 2014): Simulation results and meta data are stored in a network friendly way using the JSON format[30] where the core data is packed with msgpack[31].

4.4.1 Result meta data

The result format should store more than just numeric values. In order to interpret the results correctly, the following additional information seems especially valuable:

[25] https://openmodelica.org/doc/OpenModelicaUsersGuide-/latest/technical_details.html#the-matv4-result-file-format

[26] https://github.com/jraedler/DyMat/ https://github.com/kdavies4/ModelicaRes/

[27] https://www.hdfgroup.org/solutions/hdf5/

[28] https://github.com/ScientificDataFormat

[29] github.com/xogeny/recon

[30] https://www.json.org/json-en.html

[31] https://msgpack.org/

- Which simulation tool and of which version generated the result? Which simulation settings have been used?
- Which system model, which validated (sub-)models and which boundary conditions/scenarios were used to produce the simulation results?
- What are the interesting variables for analysis? In large Modelica result files, finding relevant variables for analysis might be difficult. Two possible ways to improve this situation could be:
  1) Predefined plots – based on figure annotations of MLSv35 (Modelica Association 2021) – could be also available in the result format.
  2) The file modelDescription.xml of FMI defines the system model interface – inputs, outputs and parameters – in a re-usable, standardized way. This interface information could also be available in result formats.
- What is the expected accuracy of the simulation? The upcoming FMI for embedded systems (eFMI) specification[32] (Lenord et al. 2021) defines tolerances.

When generating reference results for the MSL, some of this meta data is stored in a separate file creation.txt, see an example in the MSL[33].

The *SSP Traceability Specification*[34] is currently under development within the Modelica Association Project "System Structure and Parameterization of Components for Virtual System Design"[35]. The approach is based on a so-called *glue particle*, an XML file providing a consistent data schema along the simulation process. For specifics of the proposed file format, see *Simulation Task Meta Data* in file STMD.xsd. If this concept would be applied to Modelica and supported by Modelica tools, it could save a lot of today's manual documentation work.

4.4.2 Results for post-processing

Most result formats are designed in a way, that the Modelica tool can conveniently write those files during simulation. For archiving purposes, it is also important to retain the readability of the files even when the source tool is no longer available. Moreover, post-processing should also be possible in any external tool, like Excel, Julia, Matlab, Python, etc. If a standardized result file format would be available, both issues could be solved by standardized reading procedures for different languages.

5 Conclusions and outlook

The paper summarizes current challenges in Modelica-related continuous development processes with regards to modeling, simulation, data management and calibration/verification. A particular focus was given on the development and handling of Modelica models and libraries.

Several aspects have been identified that need to be improved to arrive at a reliable process for the development of credible models and digital twins based on coherent Modelica Association standards (Modelica language, FMI, SSP, DCP, eFMI)[36]. The upcoming eFMI standard (Lenord et al. 2021) goes already in the right direction by including tolerance-defined reference results in an eFMI model to support automatic tests and verification of generated production code. The glue particle approach in the SSP-project might be used for all MA standards so that tool chains from Modelica models to FMI, SSP, DCP, and eFMI components are completely traceable and no information is lost. Furthermore, additional information needs to be added to define the domain of validity of a model. It might also be necessary to add further quality measures. Simulation results need to be stored in a standardized way both for exchange between tools, as well as for archiving purposes. The recon format with glue particle information included might be considered for all Modelica Association standards and reference files.

As an overall target, the UPSIM project description states: "Enable companies to safely collaborate with simulations, in a repeatable, reliable, and robust manner, and for implementing simulation in a Credible Digital Twin setting as a strategic capability to become an important factor in quality, cost, time-to-market, and overall competitiveness." This view could become a guideline for further development of the Modelica Association standards.

Acknowledgements

This work has been partly supported by the European ITEA3 Call6 project UPSIM[37] – Unleash Potentials in Simulation (number 19006). The work was funded by the German Federal Ministry of Education and Research (BMBF, grant numbers 01IS20072H and 01IS20072G).

We would like to thank the reviewers of this paper as well as Tobias Bellmann for quite a lot of constructive improvement proposals. We would also like to thank Thomas Alpögger, Daniel Bouskela, Andreas Junghanns, Martin Krammer, Lars Mikelsons, Mugur Tatar for their comments on some details that we have taken into account.

References

Alistair, Mavin Mav and Philip Wilkinson (2019). "Ten Years of EARS". In: *IEEE Software* 36.5, pp. 10–14. DOI: 10.1109/MS.2019.2921164.

Bals, J., W. Fichter, and M. Surauer (1997). "Optimization of magnetic attitude- and angular momentum control for low earth orbit satellites". In: *Proceedings Third International Conference on Spacecraft Guidance, Navigation and Control Systems 1996*. ESTEC, Noordwijk, The Netherlands. URL:

[32] https://emphysis.github.io/pages/downloads/efmi_specification_1.0.0-alpha.4.html#definition-of-csv-data

[33] https://github.com/modelica/MAP-LIB_ReferenceResults/blob/v4.0.0/Modelica/Blocks/Examples/PID_Controller/creation.txt

[34] github.com/PMSFIT/SSPTraceability

[35] ssp-standard.org

[36] https://modelica.org/

[37] https://itea3.org/project/upsim.html

https://ui.adsabs.harvard.edu/link_gateway/1997ESASP.381..559B/ADS_PDF.

Bouskela, Daniel, Alberto Falcone, et al. (2021). "Formal Requirements Modeling for Cyber-Physical Systems Engineering: an integrated solution based on FORM-L and Modelica". In: *Requirements Engineering - accepted for publication*.

Bouskela, Daniel and Audrey Jardin (2018). "ETL: A New Temporal Language for the Verification of Cyberphysical Systems". In: *2018 Annual IEEE International Systems Conference (SysCon)*. URL: https://ieeexplore.ieee.org/document/8369502.

Bruder, Frederic and Lars Mikelsons (2020). "Towards Grey Box Modeling in Modelica". In: *Kuo CH., Lin PC., Essomba T., Chen GC. (eds) Robotics and Mechatronics. ISRM 2019. Mechanisms and Machine Science, vol 78*. DOI: 10.1007/978-3-030-30036-4_17.

Buse, Fabian and Tobias Bellmann (2021). "General Purpose Lua Interpreter for Modelica". In: *Proceedings of the 14th International Modelica Conference*.

Corso, Antony et al. (2020). "A Survey of Algorithms for Black-Box Safety Validation". In: *arXiv:2005.02979*. URL: https://arxiv.org/abs/2005.02979.

Dempsey, M. et al. (2006). "Coordinated Automotive Libraries for Vehicle System Modelling". In: *5th International Modelica Conference*. Vienna, Austria, pp. 33–41. URL: https://modelica.org/events/modelica2006/Proceedings/sessions/Session1b2.pdf.

Department of Defense (2016). *Aircraft Electric Power Characteristics (MIL-STD-704F_CHG-1)*. Tech. rep. URL: http://everyspec.com/MIL-STD/MIL-STD-0700-0799/MIL-STD-704F_CHG-1_55461/.

Elmqvist, Hilding et al. (2021). "Modia - Equation Based Modeling and Domain Specific Algorithms". In: *Proceedings of the 14th International Modelica Conference*.

Heinkel, Hans-Martin and Kim Steinkirchner (2021). *Credible Simulation Process*. Tech. rep. Robert Bosch GmbH and PROSTEP AG. URL: https://setlevel.de/neuigkeiten/credible-simulation-process.

Joos, Hans-Dieter (2015). "Application of Optimization-Based Worst Case Analysis to Control Law Assessment in Aerospace". In: *Advances in Aerospace Guidance, Navigation and Control*. DOI: 10.1007/978-3-319-17518-8_4.

König, Christian et al. (2020). "Traceability in the Model-based Design of Cyber-Physical Systems". In: *Proceedings of the American Modelica Conference 2020*. Boulder, USA, pp. 168–178. DOI: 10.3384/ecp20169168.

Kuhn, Martin R., Martin Otter, and Loic Raulin (2008). "A Multi Level Approach for Aircraft Electrical Systems Design". In: *6th International Modelica Conference*. Bielefeld, Germany, pp. 95–101. URL: https://modelica.org/events/modelica2008/Proceedings/sessions/session1d1.pdf.

Labusch, Andreas et al. (2014). "Worst Case Braking Trajectories for Robotic Motion Simulators". In: *IEEE International Conference on Robotics & Automation (ICRA)*. Hong Kong, China. DOI: 10.1109/ICRA.2014.6907333.

Leimeister, Mareike (2019). "Python-Modelica Framework for Automated Simulation and Optimization". In: *Proceedings of the 13th International Modelica Conference*. DOI: 10.3384/ecp1915751.

Lenord, Oliver et al. (2021). "eFMI: An open standard for physical models in embedded software". In: *Proceedings of the 14th International Modelica Conference*.

Mesa-Moles, L. et al. (2019). "Robust Calibration of Complex ThermosysPro Models using Data Assimilation Techniques: Application on the Secondary System of a Pressurized Water Reactor". In: *Proceedings of the 13th International Modelica Conference*. DOI: 10.3384/ecp19157553.

Modelica Association (2021-02). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.5*. Tech. rep. Linköping: Modelica Association. URL: https://specification.modelica.org/maint/3.5/MLS.pdf.

Otter, Martin et al. (2015). "Formal Requirements Modeling for Simulation-Based Verification". In: *11th International Modelica Conference*. Versailles, France, pp. 625–635. DOI: 10.3384/ecp15118625.

Pfeiffer, Andreas (2012). "Optimization Library for Interactive Multi-Criteria Optimization Tasks". In: *9th International Modelica Conference*. Munich, Germany, pp. 669–680. DOI: 10.3384/ecp12076669.

Pfeiffer, Andreas, Ingrid Bausch-Gall, and Martin Otter (2012). "Proposal for a Standard Time Series File Format in HDF5". In: *9th International Modelica Conference*. Munich, Germany, pp. 495–505. DOI: 10.3384/ecp12076495.

Reiner, M. (2011). "Modellierung und Steuerung von strukturelastischen Robotern". PhD thesis. Technische Universität München, Fakultät für Maschinenwesen.

Riedmaier, S. et al. (2021). "Unified Framework and Survey for Model Verification, Validation and Uncertainty Quantification". In: *Archives of Computational Methods in Engineering 28*, pp. 2655–2688. DOI: 10.1007/s11831-020-09473-7.

Schamai, Wladimir (2013). "Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool". PhD thesis. University of Linköping. URL: http://liu.divaportal.org/smash/record.jsf?pid=diva2:654890.

Tatar, Mugur and Jakob Mauss (2014). "Systematic Test and Validation of Complex Embedded Systems". In: *ERTS 2014 - Embedded Real Time Software and Systems*. Toulouse, France. URL: https://www.researchgate.net/publication/259871632_Systematic_Test_and_Validation_of_Complex_Embedded_Systems/.

Thuy, Nguyen (2014). "FORM-L: A Modelica Extension for Properties Modelling Illustrated on a Practical Example". In: *10th International Modelica Conference*. Lund, Sweden, pp. 1227–1236. DOI: 10.3384/ecp140961227.

Tiller, Michael and Peter Harman (2014). "recon – Web and network friendly simulation data formats". In: *Proceedings of the 10th International Modelica Conference*. DOI: 10.3384/ecp140961081.

Tundis, Andrea et al. (2017). "Model-Based Dependability Analysis of Physical Systems with Modelica". In: *Hindawi, Modelling and Simulation in Engineering, Volume 2017*. DOI: 10.1155/2017/1578043.

Wei, Wang et al. (2016). "Vibration performance analysis of vehicle with the non-pneumatic new mechanical elastic wheel in the impulse input experiment". In: *Journal of Vibroengineering, Vol. 18, Issue 6, 2016*. DOI: 10.21595/jve.2016.16988.

Zimmer, Dirk, Martin Otter, and Elmqvist (2014). "Custom Annotations: Handling Meta-Information in Modelica". In: *10th International Modelica Conference*. Lund, Sweden, pp. 174–182. DOI: 10.3384/ecp14096173.

Modeling of A Bearing Test Bench and Analysis of Defect Bearing Dynamics in Modelica

Diwang Ruan[1] Zhirou Li[2] Clemens Gühmann[1]

[1]Chair of Electronic Measurement and Diagnostic Technology, TU Berlin, Germany
diwang.ruan@campus.tu-berlin.de , clemens.guehmann@tu-berlin.de
[2]School of Electronic Engineering and Computer Science, TU Berlin, Germany , lizhirou0423@hotmail.com

Abstract

In data-driven bearing fault diagnosis, sufficient fault data is fundamental for algorithm training and validation, however, in most industry applications, only very few fault measurements can be provided, which brings bearing dynamics model as an alternative to produce bearing response under defects. In this paper, a Modelica model for the whole bearing test rig was built, including test bearing, driving motor and hydraulic loading system. For the test bearing, a 5 degree-of-freedom (5-DoF) model was proposed to identify the normal bearing dynamics, and a fault model was employed to characterize the defect position, defect size, defect shape and multiple defects. Theory and process to implement the virtual bearing test bench in Modelica were detailed, and 3 cases were conducted to validate the effectiveness of the proposed model.

Keywords: Bearing Diagnosis, Fault Modeling, Modelica, Bearing Test Bench

1 Introduction

Dynamics simulation under defect is essential for bearing fault diagnostics. However, traditional research method based on experiments is of high cost and low efficiency since it requires a real test rig and the defect needs to be generated artificially. Furthermore, out of safety consideration, experiment-based research usually runs under only some specific working conditions and defect sizes, which restricts the exploration of bearing dynamics under extreme conditions and fault specifications. To bridge the gap, this paper proposes a virtual bearing test bench in Modelica to serve as a general platform for fault bearing dynamics simulation.

To date, the methods for fault bearing dynamics simulation can be classified into 2 categories, namely mechanism-based models and signal-based models. Cui et al. (Cui, X. Chen, and S. Chen (2015)) built a 5-DoF model to characterize bearing's dynamic behavior. Besides, a defect model was also established to deal with defect position, defect shape and defect size (Liu, Shao, and Lim (2012)). Whereas, other researchers investigated the fault bearing dynamics response from the perspective of signal analysis. The first model identifying the amplitude spectrum of bearing with a single defect on the inner race was proposed by McFadden in 1983 (McFadden and Smith (1984)). In 2000, slight random variations were further incorporated into the impulse responses to resemble actual vibration signals caused by bearing faults (Ho and R. Randall (2000)). After that, Cong et al. (Cong et al. (2013)) put forward a new fault signal model for bearing based on the combination of decaying oscillation fault signal model and rotor dynamic response influence, especially, the defect load was divided into alternate load and determinate load.

For real test bench, dynamics from driving and loading systems also affect test bearing response. Nevertheless, to our exhausted knowledge, nearly all published papers on bearing fault modeling only focus on the bearing and just set speed and load as constants, without considering the dynamics from driving motor and loading actuator, which leads to the goal to build a whole bearing test bench in this paper.

The remainder of this paper is organized as follows. Section 2 details the modeling theory of bearing test bench, including 5-DoF bearing dynamics model, bearing defect model, driving and loading system model. Section 3 outlines the model structure in Modelica and demonstrates how to use this virtual test bench to simulate bearing with specific defects. Section 4 concludes this paper.

2 Modeling in Modelica

2.1 Test Bearing

The ball bearing is composed of outer ring, inner ring, cage and rolling elements. A normal bearing achieves dynamic balance in a stable operating condition, while a series of impulses will be generated once there is a defect between the contact surfaces. In the following, a 5-DoF dynamics model and a defect model will be introduced.

2.1.1 5-DoF Dynamics Model

This model describes the nonlinear dynamic behavior of bearing, as shown in Figure 1. In the 5-DoF model, 4 DoF represents the horizontal and vertical direction of inner and outer rings, and 1 DoF stands for the vertical direction of a unit resonator, which is modeled as spring-mass system (Cui, X. Chen, and S. Chen (2015)).

Based on Newton's second law, the bearing dynamic

DOI
10.3384/ecp21181373

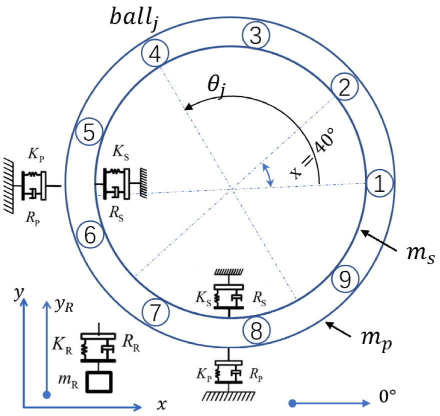


Figure 1. The 5-DoF model of bearing (Cui, X. Chen, and S. Chen (2015)).

equilibrium equations can be formulated as Equation 1 (Cui, X. Chen, and S. Chen (2015)).

$$\begin{aligned} & m_s\ddot{x_s}+R_s\dot{x_s}+K_sx_s+f_x=0\ , \\ & m_s\ddot{y_s}+R_s\dot{y_s}+K_sy_s+f_y=F_y-m_sg\ , \\ & m_p\ddot{x_p}+R_p\dot{x_p}+K_px_p-f_x=0\ , \\ & m_p\ddot{y_p}+(R_p+R_R)\dot{y_p}+(K_p+K_r)y_p-R_R\dot{y_b}\ , \\ & -K_Ry_b-f_y=-m_pg\ , \\ & m_R\ddot{y_b}+R_R(\dot{y_b}-\dot{y_p})+K_R(y_b-y_p)=-m_Rg\ . \end{aligned} \tag{1}$$

f_x and f_y are contact force at x and y axis respectively, F_y is external load. The meanings and values of other variables are summarized in Table 1. According to Hertzian contact theory, the contact force between rolling element and raceways can be given by:

$$f_j=K_b\delta_j^{1.5}\ , \tag{2}$$

with j from 1 to n_b, n_b is the number of rolling elements. K_b stands for ball's stiffness, δ denotes deformation. The deformation of the j^{th} ball, δ_j, is determined by the displacement between the inner and outer races, the angular position θ_j and the total clearance c caused by the oil film and assembly clearance, as:

$$\delta_{\mathrm{raw}_j}=(x_s-x_p)\cos\theta_j+(y_s-y_p)\sin\theta_j-c\ . \tag{3}$$

The angular position of the j^{th} ball can be calculated by Equation 4.

$$\theta_{\mathrm{raw}_j}=\frac{2\pi(j-1)}{n_b}+\omega_ct+\phi_0\ , \tag{4}$$

where ϕ_0 is initial cage angular position and ω_c is cage angular frequency, which can be further obtained from shaft frequency ω_s like Equation 5.

$$\omega_c=\left(1-\frac{D_b}{D_p}\right)\frac{\omega_s}{2}\ , \tag{5}$$

where D_b and D_p are the ball diameter and pitch diameter respectively.

Normally, for bearings in real applications, there exists inevitable sliding when a ball rolls on the raceways. The sliding direction depends on where the ball is located, when the ball enters into the load zone, the angular speed of the ball center is faster than that of the cage, otherwise, the ball slides backward. Consequently, when sliding considered , the angular position of each ball can be modified by Equation 6 (Cui, X. Chen, and S. Chen (2015)).

$$\theta_j=\theta_{\mathrm{raw}_j}+\xi_j\left(\frac{1}{2}rand\right)\phi_{\mathrm{slip}}\ . \tag{6}$$

There are two constants and a sign function in Equation 6. ϕ_{slip} is a parameter defining the mutation percentage of average contact frequency, which is normally between 0.01 and 0.02 rad. *rand* is a random number with uniform distribution in the range of $[0,1]$, and the sign function ξ_j is expressed as:

$$\xi_j=\begin{cases}1, & \text{load zone}\\ -1, & \text{else}\end{cases} \tag{7}$$

Considering δ_j should be nonnegative in physics, thus, its final value is determined by:

$$\delta_j=Max(\delta_{\mathrm{raw}_j},0)\ . \tag{8}$$

With the contact force of each ball obtained from Equation 2, the total contact forces in x and y direction can be determined with the Equations 9 and 10.

$$f_x=\sum_{j=1}^{n_b}f_j\cos\theta_j\ , \tag{9}$$

$$f_y=\sum_{j=1}^{n_b}f_j\sin\theta_j\ . \tag{10}$$

2.1.2 Defect Model

When bearing has defects either on races or balls, an additional deformation, δ_{fau}, will release when ball moves over the defect zone. Thus, with defect considered, deformation of the j^{th} ball can be further identified as:

$$\delta_{\mathrm{raw}_j}=(x_s-x_p)\cos\theta_j+(y_s-y_p)\sin\theta_j-c-\delta_{fau_j}\ . \tag{11}$$

Once bearing deformation under defect is obtained, it can be substituted into Equations 9 and 10, where the nonlinear contact force can be calculated and further substituted into Equation 1 to get the fault bearing response. Apparently, δ_{fau} changes with defect position, defect shape and number of defects, which will be discussed respectively in the following.

2.1.3 Defect Position

Firstly, four basic geometrical parameters are chosen to characterize the defect, as demonstrated in Figure 2, the defect width B, the defect depth H_d, the defect initial angle ϕ_d and the defect span angle $\Delta\phi_d$. Take a defect on the outer ring as an example, the relation between $\Delta\phi_d$ and B can be expressed as:

$$\sin\left(\frac{1}{2}\Delta\phi_d\right) = \frac{B}{D_b + D_p} \ . \tag{12}$$

Suppose the local defect depth is c_d, the additional deformation δ_{fau} generates only when a ball falls into the defect zone within ϕ_d and $\phi_d + \Delta\phi_d$, so the deformation released by defect on raceway is given by:

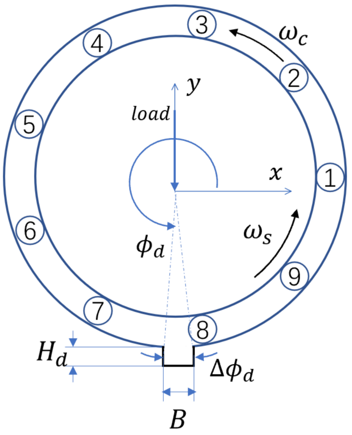


Figure 2. Size definition of defect on the outer ring.

$$\delta_{fau_j} = \begin{cases} c_d, & \phi_d \le \theta_j \le \phi_d + \Delta\phi_d \\ 0, & else \end{cases} \tag{13}$$

The defect location on the outer ring or inner ring changes with different rules. For the outer ring, the defect is fixed at the defect initial angle ϕ_{do}, however, for the inner ring, the defect location changes with time when the inner ring rotates. Thus, ϕ_d in Equation 13 can be further modeled as follows.

$$\phi_d = \begin{cases} \phi_{do}, & \text{defect on the outer ring} \\ \omega_s t + \phi_{di}, & \text{defect on the inner ring} \end{cases} \tag{14}$$

Different from rings, when a defect happens on a rolling element, the defect spins with ball speed ω_b and its position ϕ_s can be obtained like:

$$\phi_s = \omega_b t + \phi_{s_{ini}} \ , \tag{15}$$

hereby the ball speed ω_b can be calculated from shaft speed as follows,

$$\omega_b = \frac{\omega_s}{2}\frac{D_p}{D_b}\left[1 - \left(\frac{D_b}{D_p}\cos\alpha\right)^2\right] \ . \tag{16}$$

The defect on balls contacts the inner and outer ring periodically. Besides, the curvature radiuses of inner and outer rings are different, therefore, the same defect span angle $\Delta\phi_d$ produces different angular widths. The angular widths of defect on the outer ring and inner ring, $\Delta\phi_{bo}$ and $\Delta\phi_{bi}$, can be calculated by Equations 17 and 18.

$$\Delta\phi_{bo} = \Delta\phi_d \frac{D_b}{Do} \ , \tag{17}$$

$$\Delta\phi_{bi} = \Delta\phi_d \frac{D_b}{Di} \ , \tag{18}$$

with D_o and D_i as the diameters of outer ring and inner ring respectively.

Obviously, the curvature radius determines the depth when ball enters into the raceway, and the curvature radiuses of ball c_{dr}, inner ring c_{di} and outer ring c_{do} can be obtained respectively by Equations 19, 20 and 21 (Mishra, Samantaray, and Chakraborty (2017)).

$$c_{dr} = \frac{1}{2\left(D_b - \sqrt{D_b^2 - B^2}\right)} \ , \tag{19}$$

$$c_{di} = \frac{1}{2\left(D_i - \sqrt{D_i^2 - B^2}\right)} \ , \tag{20}$$

$$c_{do} = \frac{1}{2\left(D_o - \sqrt{D_o^2 - B^2}\right)} \ , \tag{21}$$

During one revolution of the ball, defect contacts the inner ring and outer ring in succession, with an angular distance of π. So, c_d can be given by Equation 22 (Mishra, Samantaray, and Chakraborty (2017)).

$$c_d = \begin{cases} c_{dr} - c_{do}, & 0 \le \varphi_s \le \Delta\phi_{bo} \\ c_{dr} + c_{di}, & \pi \le \varphi_s \le \pi + \Delta\phi_{bi} \end{cases} \tag{22}$$

Deformation released by fault only appears on the fault ball (k^{th} ball). As a result, the contact deformation with ball defect is given by:

$$\delta_j = \begin{cases} 0, & j \ne k \\ c_d, & j = k \end{cases} \tag{23}$$

2.1.4 Multiple Defects

When bearing has multiple defects, the model is expanded into a matrix. For simplicity, no matter how many and what kind of defects the bearing has, the total impact on the bearing vibration is supposed to be the sum of effect that each defect has on each ball. Thus, a $3 \times n_d$ matrix N is defined to deal with multiple defects modeling.

$$N_{3\times n_d} = \begin{bmatrix} p_1 & p_2 & \cdots & p_{n_d} \\ \phi_{d_1} & \phi_{d_2} & \cdots & \phi_{d_{nd}} \\ \Delta\phi_{d_1} & \Delta\phi_{d_1} & \cdots & \Delta\phi_{d_{nd}} \end{bmatrix} \tag{24}$$

where p_{n_d}, $\phi_{d_{n_d}}$ and $\Delta\phi_{d_{n_d}}$ stand for the n_d^{th} defect position, defect initial angle and defect span angle respectively. In this model, p_i is defined with Equation 25 for

$i = 1, 2 \cdots n_d$.

$$p_i = \begin{cases} 1, & \text{outer ring fault} \\ 2, & \text{inner ring fault} \\ 3, & \text{ball fault} \end{cases} \tag{25}$$

The deformation on each ball caused by each defect can be determined based on above discussion. For a bearing with n_b rolling elements and n_d defects, the deformation depth is an $n_d \times n_b$ matrix as:

$$\delta_{n_d \times n_b} = \begin{bmatrix} \delta_{11} & \cdots & \delta_{1n_b} \\ \vdots & \ddots & \vdots \\ \delta_{nd_1} & \cdots & \delta n_d n_b \end{bmatrix} \tag{26}$$

The total deformation depth caused by all defects for each ball is obtained through elementwise addition in column. Therefore, a deformation vector with n_b dimensions is given as Equation 27.

$$\delta_{1 \times n_b} = \left[\delta_1, \delta_1, \cdots, \delta_{n_b}\right] \tag{27}$$

Consequently, the defect model for bearing with n_d defects is established. The calculation of contact force and acceleration is the same as the single defect model.

2.1.5 Defect Shape and Size

Besides defect position, defect shape also has much influence on bearing acceleration response. Most existing bearing defect models, however, simplify the defect depth c_d as a constant value. In practice, the value of c_d differs with defect shape as well as the ratio of defect size to ball diameter (Cui, X. Chen, and S. Chen (2015)). In order to accurately model the defect shape, the released deformation is modeled by a piece-wise function. In this research, two ratios, ball to defect ratio η_{bd} and length to width ratio η_d, are defined in Equation 28 and 29 for defect shape modeling,

$$\eta_{bd} = \frac{D_b}{min(L, B)}, \tag{28}$$

$$\eta_d = \frac{L}{B}, \tag{29}$$

where B and L represent the width and length of the defect respectively (Cui, X. Chen, and S. Chen (2015)). With the combination of these two ratios, the defect shape is grouped into four types (H_{r1}, H_{r2}, H_{r3}, H_{r4}), which is given by Equation 30 (Cui, X. Chen, and S. Chen (2015)).

$$c_d = \begin{cases} H_{r1}, & \eta_{bd} \gg 1 \\ H_{r2}, & \eta_{bd} > 1 and \eta_d \leq 1 \\ H_{r3}, & \eta_{bd} > 1 and \eta_d > 1 \\ H_{r4}, & \eta_{bd} \leq 1 \end{cases} \tag{30}$$

The maximal depth of defect for each case is denoted with $c_d^{'}$, which will be explained later.

In case 1, the defect size is too small compared with ball diameter, so the ball leaves defect immediately as soon as it contacts the defect. Therefore, as shown in Figure 3(a), there is no deformation change in the defect zone, and H_{r1} can be modeled as:

$$H_{r1} = c_d^{'}, \qquad 0 \leq \varphi_j \leq \Delta\phi_d. \tag{31}$$

For case 2, the ball's moving path over the defect zone is like a half-sine wave, as shown in Figure 3(b). The deformation released from defect increases gradually to the maximum and then decreases, which can be given by:

$$H_{r2} = c_d^{'} \sin\left(\frac{\pi}{\Delta\phi_d}\varphi_j\right), \qquad 0 \leq \varphi_j \leq \Delta\phi_d \tag{32}$$

In case 3, as revealed in Figure 3(c), c_d rises up to the maximal depth gradually and remains at the maximum between ϕ_1 and ϕ_2. After that, c_d begins to decrease when the ball gets out of defect. This shape is expressed by Equations 33 and 34.

$$H_{r3} = \begin{cases} c_d^{'} \sin\left(\frac{\pi}{2\phi_1}\varphi_j\right) & 0 \leq \varphi_j < \phi_1 \\ c_d^{'} & \phi_1 \leq \varphi_j < \phi_2 \\ c_d^{'} \sin\left(\frac{\pi}{2\phi_1}\varphi_j + \frac{\pi}{2}\right) & \phi_2 \leq \varphi_j < \Delta\phi_d \end{cases} \tag{33}$$

$$\phi_1 = \Delta\phi_d \lambda, \quad \phi_2 = \Delta\phi_d(1 - \lambda), \tag{34}$$

hereby λ is the ratio of ϕ_1 to $\Delta\phi_d$. ϕ_1 is the position where the ball reaches defect bottom.

In case 4, since the ratio of ball to defect η_{bd} is less than that in case 3. Thus, the ball reaches the defect bottom within a complete 1/4 sine wave, and defect shape can be modeled as:

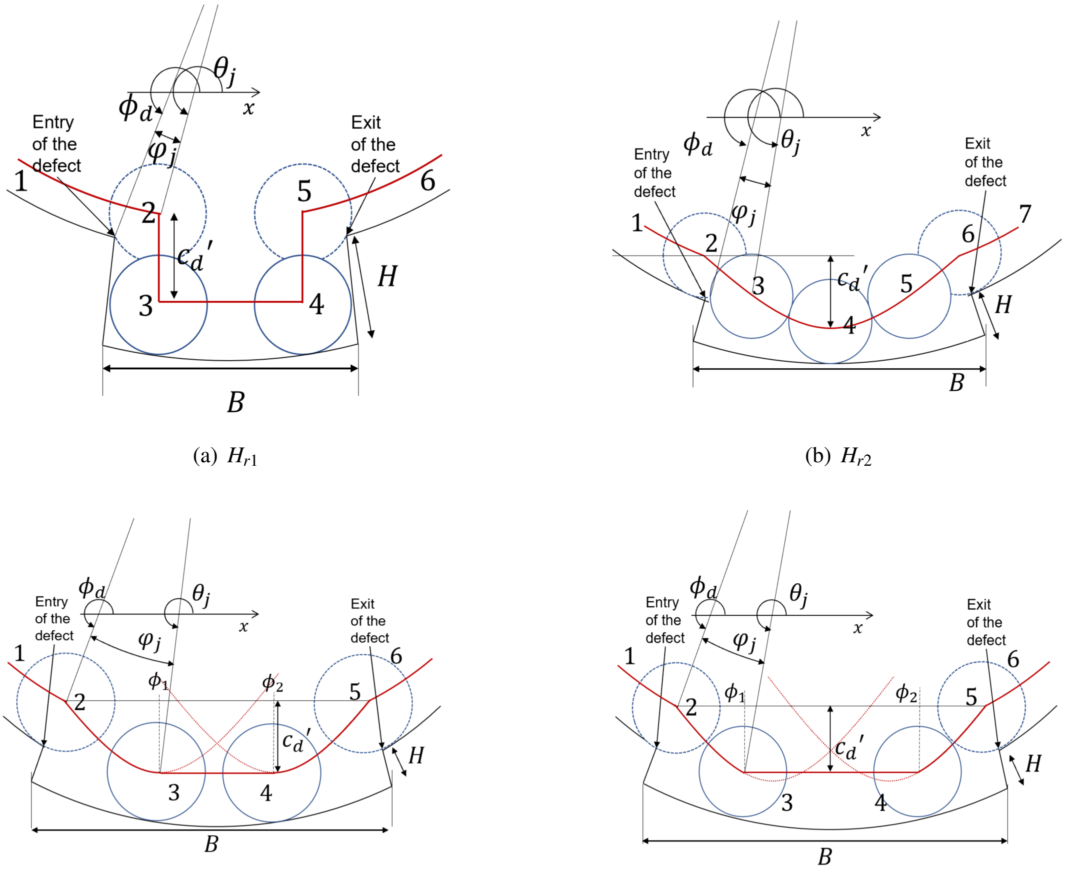
$$H_{r4} = \begin{cases} c_d^{'} \sin\left(\frac{2\pi}{\Delta\phi_d}\varphi_j\right), & 0 \leq \varphi_j < \phi_1 \\ c_d^{'}, & \phi_1 \leq \varphi_j < \phi_2 \\ c_d^{'} \sin\left(2\pi - \frac{2\pi}{\Delta\Phi_d}\varphi_j\right), & \phi_2 \leq \varphi_j < \Delta\phi_d \end{cases} \tag{35}$$

The maximum depth $c_d^{'}$ is given by Equations (Cui, X. Chen, and S. Chen (2015)):

$$H_d = \frac{D_b}{2} - \sqrt{\left(\frac{D_b}{2}\right)^2 - \left(\frac{B}{2}\right)^2}, \tag{36}$$

with

$$c_d^{'} = min(H, H_d). \tag{37}$$



(a) H_{r1}

(b) H_{r2}

(c) H_{r3}

(d) H_{r4}

Figure 3. Defect depth c_d under different defect shape.

Table 1. Bearing parameters.

Symbol	*Quantity*	*Value*
m_s	Shaft mass	$3{,}263\,8\mathrm{kg}$
R_s	Shaft damping	$1{,}376\,8 \cdot 10^3\mathrm{N\,s\,m^{-1}}$
K_s	Shaft stiffness	$7{,}42 \cdot 10^7\mathrm{N\,m^{-1}}$
m_p	Pedal mass	$6{,}638\mathrm{kg}$
R_p	Pedal damping	$2{,}210\,7 \cdot 10^3\mathrm{N\,s\,m^{-1}}$
K_p	Pedal stiffness	$1{,}51 \cdot 10^7\mathrm{N\,m^{-1}}$
m_R	Resonator mass	$1\mathrm{kg}$
R_R	Resonator stiffness	$9{,}424\,8 \cdot 10^3\mathrm{N\,s\,m^{-1}}$
K_R	Resonator stiffness	$8{,}882\,6 \cdot 10^9\mathrm{N\,m^{-1}}$
n_b	Ball number	9
D_p	Pitch diameter	$3{,}932 \cdot 10^{-2}\mathrm{m}$
D_b	Ball diameter	$7{,}94 \cdot 10^{-3}\mathrm{m}$
ϕ_{slip}	Ball slip angle	0,01 rad
rand	Mutation percentage	0
K_b	Ball stiffness	$1{,}89 \cdot 10^{10}\mathrm{N\,m^{-1}}$
c	Bearing clearance	0
α	Contact angle	0°

Table 2. Defect parameters.

Symbol	*Quantity*	*Value*
L	Axial defect length	$3 \cdot 10^{-4}\mathrm{m}$
B	Race defect width	$10 \cdot 10^{-4}\mathrm{m}$
H	Radial defect depth	$6 \cdot 10^{-4}\mathrm{m}$
λ	Ratio of ϕ_1 to $\Delta\phi_d$	0,2
ϕ_d	Initial defect position	270°
k	Order of the defective ball	4
w	Spall width	$3 \cdot 10^{-3}\mathrm{m}$
$\phi_{s_{ini}}$	Initial position of spall	0°

2.1.6 Parameters

Parameter specification of the 5-DoF dynamics model is shown in Table 1, which includes geometrical and material parameters (Mishra, Samantaray, and Chakraborty (2017)). The defect model parameters can be defined by users out of simulation requirements. In this paper, they are set as in Table 2.

Once the bearing model has been finished, the characteristic frequencies can be calculated by Equations 38-42, including ball pass frequency of outer ring ($BPFO$), ball pass frequency of inner ring ($BPFI$), ball spin frequency (BSF), fundamental train frequency (FTF) and element defect frequency (EDF). All of them will be used for analysis and discussion in the following sections (Robert B Randall and Antoni (2011)).

$$BPFO = \frac{n_b f}{2}\left(1 - \frac{D_b}{D_p}\cos\alpha\right) \; , \tag{38}$$

$$BPFI = \frac{n_b f}{2}\left(1 + \frac{D_b}{D_p}\cos\alpha\right) \; , \tag{39}$$

$$BSF = \frac{f D_p}{2 D_b}\left[1 - \left(\frac{D_b}{D_p}\cos\alpha\right)^2\right] \; , \tag{40}$$

$$FTF = \frac{f}{2}\left(1 - \frac{D_b}{D_p}\cos\alpha\right) \; , \tag{41}$$

$$EDF = 2BSF \; . \tag{42}$$

2.2 Driving System

Besides test bearing, the virtual bearing test bench also consists of a driving module and a loading module, where the loading module guarantees that test bearing works under defined external load, and driving module is responsible for speed profile definition. In this paper, the driving module is modeled by a DC motor and a shaft, while the loading module is modeled by an electro-hydraulic servo system.

Based on Newton's second law and Kirchhoff's voltage law, the DC motor can be modeled as:

$$J_M \frac{d\omega}{dt} + C_n \omega + T_L = T_e \; , \tag{43}$$

$$L_M \frac{di}{dt} + Ri + e = U \; , \tag{44}$$

where the generated torque T_e and the back electromotive force (EMF) e can be further modeled as following.

$$T_e = K_t \cdot i \; , \tag{45}$$

$$e = K_e \cdot \omega \; . \tag{46}$$

The speed of the DC motor is controlled by a PID controller.

2.3 Shaft

The connecting shaft is employed to transmit the moment T_e from DC-motor. Its dynamics is modeled as:

$$k_S \int (\omega_{in} - \omega_{out})\, dt + c_S(\omega_{in} - \omega_{out}) = T_e - T_L \; , \tag{47}$$

where k_S is the stiffness and c_S is the damping coefficient, ω_{in} and ω_{out} are the input and output speed of the shaft, T_L is load torque.

2.4 Loading System

2.4.1 Components of Electro-hydraulic Servo System

The electro-hydraulic servo system consists of three components: servo amplifier, servo valve and actuator. The servo amplifier is used to convert signal from voltage to current, the servo valve is a proportional relief valve, and the actuator is a hydraulic cylinder. The asymmetrical cylinder is controlled by a four-way valve with position feedback.

2.4.2 Modeling of Electro-hydraulic Servo System

The servo amplifier is modeled as a proportional component, which amplifies the control voltage and then converts into current i to control the electromagnetic force acting on the valve spool,

$$i(t) = K_f u_e(t) \; . \tag{48}$$

The servo valve is modeled as a second-order system like follows (Rydberg (2016)).

$$G_{sv}(s) = \frac{Q_L(s)}{I(s)} = \frac{K_{gy}}{\frac{s^2}{\omega_{sv}^2} + \frac{2\xi_{sv}}{\omega_{sv}} s + 1} \; . \tag{49}$$

The gain is obtained from

$$K_{sv} = \frac{K_{IE}}{K_{SF}} \cdot K_Q \; , \tag{50}$$

where K_{IF}, K_{SF} and K_q are the gain of current to force, the stiffness of valve spool and the flow gain respectively. The natural frequency and servo valve damping coefficient are given by:

$$\omega_{sv} = \sqrt{K_{sF}/m} \; , \tag{51}$$

$$\xi_{sv} = \frac{B_p}{2\sqrt{m K_{SF}}} \; . \tag{52}$$

The actuator in this study is modeled as an asymmetric cylinder, which can be regarded as a valve-controlled piston with position feedback. In general, three sub-models are developed to describe the flow characteristics, flow balance and force balance respectively. The flow characteristics after linearization can be simplified as:

$$Q_L(s) = K_q X_v(s) - K_c P_L(s) \; , \tag{53}$$

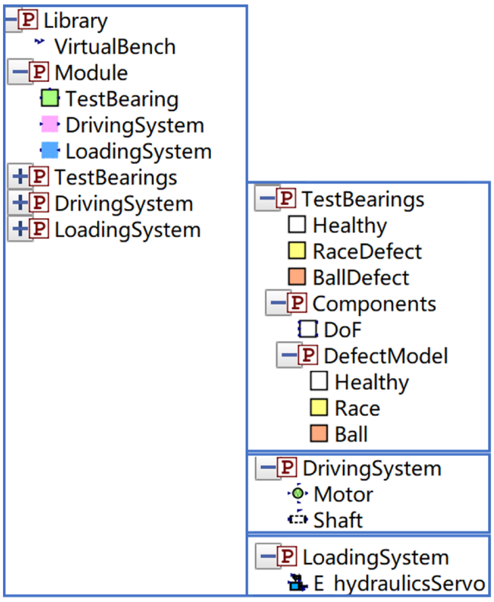
the flow balance equation is identified as:

$$Q_L(s) = A_p s X_p(s) + \lambda_c P_L(s) + \frac{v_t}{4\beta_e} s P_L(s) \; , \tag{54}$$

and the force balance equation can be formulated as:

$$A_p P_L(s) = M_t s^2 X_p(s) + B_c X_p(s) + F_L \; , \tag{55}$$

where X_v and K_c are the displacement of valve and the flow-pressure factor. Q_L, X_p and P_L stand for the load flow, the displacement of piston and the pressure of the load



Library
VirtualBench
Module
TestBearing
DrivingSystem
LoadingSystem
TestBearings
DrivingSystem
LoadingSystem

TestBearings
Healthy
RaceDefect
BallDefect
Components
DoF
DefectModel
Healthy
Race
Ball

DrivingSystem
Motor
Shaft

LoadingSystem
E hydraulicsServo

Figure 4. Structure of the developed library in OpenModelica.

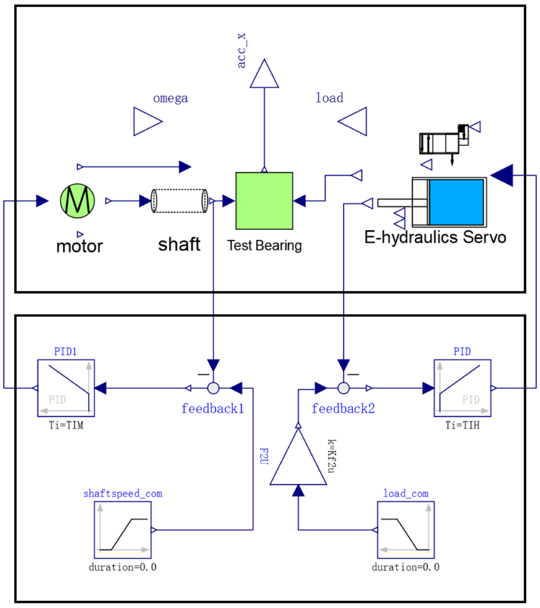


Figure 5. Layer diagram of the VirtualBench.

flow respectively. A_p, λ_c, V_t and β_e are the piston area, the total leakage coefficient, the cylinder volume as well as the volume elastic modulus coefficient. M_t and B_c are the mass of piston and the damping factor of cylinder.

Combining Equations 53 to 55 gives the transfer function of piston displacement to valve displacement, as shown in Equation 56, where the total leakage λ_c is omitted and the damping factor is assumed to be small (Mitianiec and Bac (2011)).

$$G_h(s) = \frac{X_p(s)}{X_v(s)} = \frac{K_q/A_p}{s\left(\frac{s^2}{\omega_h^2} + \frac{2\xi_h}{\omega_h}s + 1\right)}\,, \tag{56}$$

with

$$\omega_h = \sqrt{\frac{4\beta_e A_p^2}{V_t M_t}}\,, \tag{57}$$

$$\xi_h = \frac{K_c}{A_p}\sqrt{\frac{\beta_e M_t}{V_t}}\,. \tag{58}$$

In short, a proportional system is used to model the amplifier, G_{sv} for the servo valve and G_h for hydraulic cylinder. The loading system is also controlled by a PID controller.

3 Implementation

In this research, a model library is created for a virtual bearing test bench in OpenModelica-v1.16.0. As shown in Figure 4(left), 3 main modules like TestBearings, DrivingSystem and LoadingSystem are packaged and can be used as plug-in components in modeling. Figure 4(right) displays the components used in each module, and they are also sub-packaged with corresponding names. Like the TestBearings package provides three instance models as Healthy, RaceDefect and BallDefect and a Components sub-package containing a DoF model and another sub-package named DefectModel.

These models can be constructed into any configuration as required, all components, sub-packages and models can be used separately or in combination to meet user demand. Nevertheless, a configuration instance, VirtualBench, is provided at the top of Library.

3.1 System Configuration

Figure 5 demonstrates the layer diagram of proposed test bench, which consists of physical part (top) and controller part (bottom). The physical part is established with models developed in above sections, and it outputs three signals, namely rotational speed “omega”, acceleration “acc_x” and radial load “load”. Moreover, the rotational speed and radial load are inputs to the controller part to provide required conditions for test bearing. The TestBearing model has all parameters mentioned in Section 2.1, with four pages of parameter-input dialog box for users to define fault position, design parameters, material parameters, and defect parameters. Specifically, “position” is a selection parameter to define where the defect is located; design parameters include basic geometrical information such as ball number, pitch diameter and ball diameter; material properties include mass, stiffness, and damping factor of the outer ring, inner ring, rolling element as well as the resonator. Besides, the most important parameters are defect properties. Different parameters are required for different defect scenarios. As long as the number of defects and other defect parameters are given, this defect model can be used for multiple defects as well. The operating conditions are provided by Motor, Shaft, and E_hydraulicsServo. Generally, the TestBearing model can be employed to study the vibration response of fault bearing and all the parameters of any model or com-

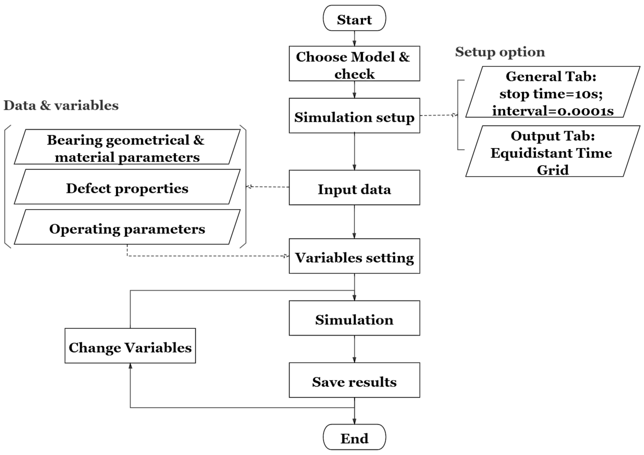


Figure 6. Flowchart of simulation with the virtual bearing test bench.

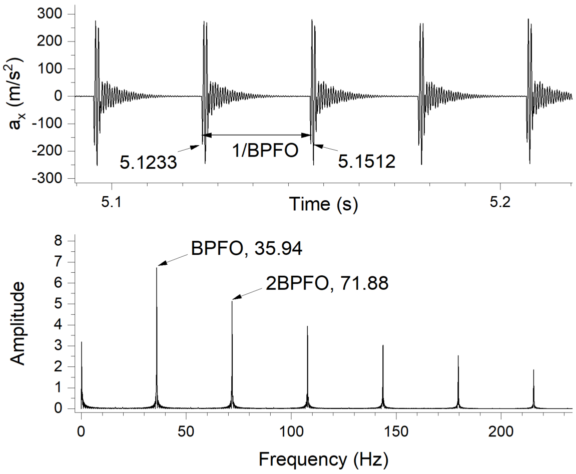


Figure 7. Response in time-domain and envelope spectrum of bearing with outer race fault.

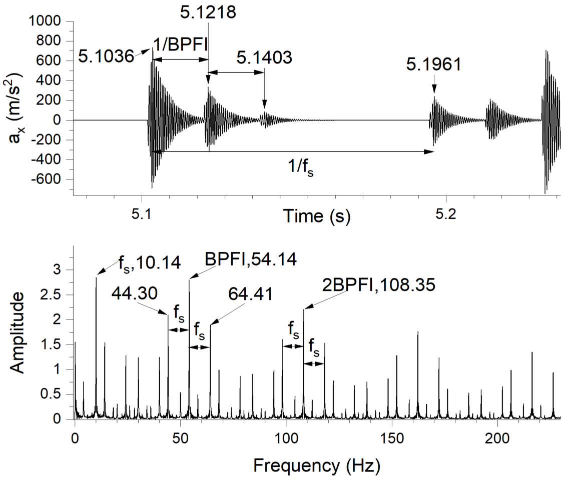


Figure 8. Response in time-domain and envelope spectrum of bearing with inner race fault.

ponent can be defined by users for specific objectives.

3.2 Procedure of Virtual Bearing Test Bench

The flowchart in Figure 6 demonstrates the process to run simulations with proposed virtual bearing test bench. Firstly, the model configuration and precheck is required, specific simulation model should be constructed based on the developed Modelica components. After that, it is necessary to set simulation period, interval length and the output format. Then, in the step of Input data, geometrical and material parameters, defect properties and operating conditions should be defined, and some parameters can be selected as variables to study the effects of defects on vibration from different aspects. At last, run the simulation and save results.

3.3 Case Simulation and Analysis

Based on the developed bearing test bench, three simulation cases are conducted for validation. Case A focuses on the defect position, case B and case C deals with multiple defects and defect shape respectively. Simulation time and interval length of 3 cases are set as 10 s and 0.0001 s, with "DASSL" solver as the integration method.

3.3.1 Case A: Defect Position

In case A, three simulations are designed to obtain the bearing responses when a defect occurs on the outer ring, inner ring or a ball respectively. The signal characteristics in both time-domain and frequency-domain are analyzed.

Figure 7 shows the time domain response and envelope spectrum of bearing with a single defect on the outer ring. The theoretical fault characteristic frequency ($BPFO$) is 35.91 Hz, corresponding to 0.0278 s. In time domain, the impulse decaying oscillation repeats with a period of 0.0279 s, and the impulse magnitude is nearly constant. In frequency domain, $BPFO$ (35.94 Hz) is extracted in the envelope spectrum, which is very close to the theoretical value.

Figure 8 describes the simulated signal when a defect is defined on the inner ring. The shaft frequency (f_s) is set as 10 Hz, so the theoretical $BPFI$ is 54.09 Hz (0.018 s). The vibration response from 5.1036 s to 5.1961 s represents the output during a whole revolution of inner ring, with 5.1036 s - 5.1403 s standing for the load zone and 5.1403 s - 5.1961 s the non-load zone. The time interval between 5.1036 s and 5.1961 s is 0.0925 s, which corresponds to f_s (10.14 Hz) in the envelope spectrum. Within one cycle, there are three peaks at 5.1036 s, 5.1218 s and 5.1403 s, and every two adjacent peaks are 0.018 s apart away, which is related to the $BPFI$. Furthermore, the inner ring defect rotates with time, which results in load change at the defect position. Thus, the signal presents various amplitudes during one cycle.

The vibration response of bearing with a defect on the ball is demonstrated in Figure 9. In time-domain, the time interval between every two impulses is approximate 0.021 s. When a defect occurs on a ball, the defect strikes both the outer ring and inner ring in a full rotation. As a result, peaks can be found in frequency-domain at EDF and its

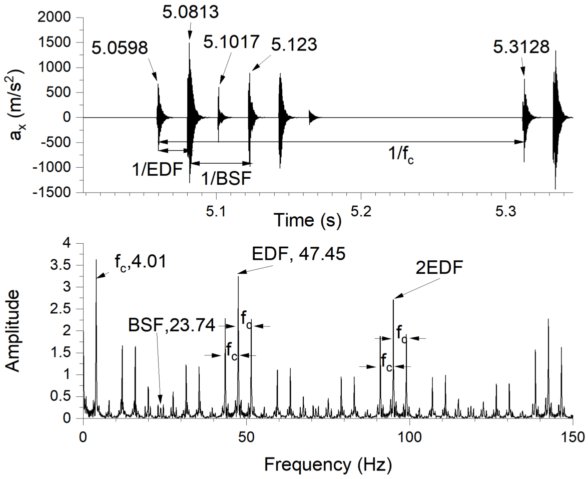


Figure 9. Response in time-domain and envelope spectrum of bearing with ball fault.

harmonics, with f_c as the sideband. The theoretical EDF is 47.50 Hz, and the simulated value is 47.45 Hz.

In short, both time-domain and frequency-domain responses contain useful defect information. In time-domain, defect position (outer ring, inner ring or ball) can be deduced from the time interval between adjacent peaks. In frequency-domain, defect position can be inferred from the characteristic frequencies ($BPFO$, $BPFI$ or EDF) in the envelope spectrum. Furthermore, the amplitude of peaks under different fault positions varies accordingly. The peak amplitudes are nearly constant when a defect occurs on the outer ring, however, the peak amplitudes change during a cycle when a defect happens on the inner ring or a ball. In addition, in the inner-ring defect, f_s can be found in the envelope spectrum as sideband, while in the ball defect, the sideband is replaced by f_c.

3.3.2 Case B: Multiple Defects

The second case focuses on multiple defects. The angle between two adjacent defects is defined as ψ and the angle between every two rolling elements in this study is 40°. Therefore, in total, there are 3 relations: $\psi > 40°$, $\psi < 40°$, $\psi = 40°$. Given space limitation, only the case with 2 defects and $\psi < 40°$ is simulated.

Two defects are defined at 255° and 285°. Once the rolling elements rotate, each ball collides with these two defects successively, resulting in two sequences of collisions. Therefore, in Figure 10, two impacts are observed in a cycle, which identifies the number of defects. According to the direction of acceleration, the impacts at 5.0712 s (B) and 5.0990 s (D) are caused by the defect at 255°, while the impacts at 5.0642 s (A) and 5.0920 s (C) by defect at 285°.

The time delay between two strikes due to multiple defects on the races can be calculated as follows (Patel, Tandon, and Pandey (2014)).

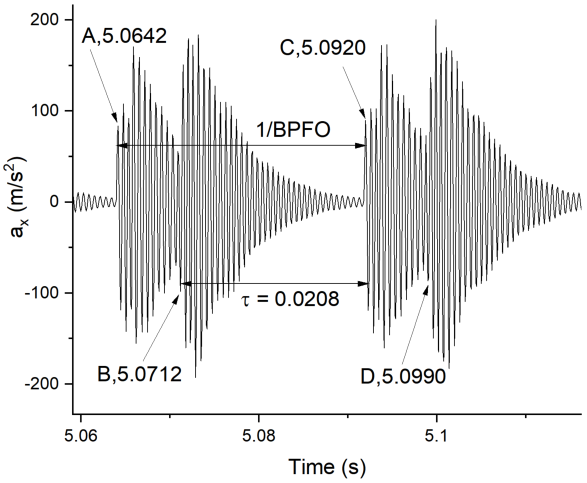


Figure 10. Bearing response with two defects on the outer ring separated by 30°.

Table 3. Time delay of two defects.

Φ	30°
τ calculated [s]	0.0209
τ simulated [s]	0.0208

$$\tau(\Phi) = \begin{cases} \dfrac{\Phi}{xf_t}, & x \leq \Phi \\ \dfrac{abs(x-\Phi)}{xf_t}, & x > \Phi \end{cases} \tag{59}$$

with

$$f_t = \begin{cases} BPFO, & \text{defects on outer ring} \\ BPFI, & \text{defects on inner ring} \end{cases} \tag{60}$$

The time delay between B and C is 0.0208 s, which corresponds to the angle between 255° and 285°. The theoretical time delay and simulation result are summarized in Table 3.

3.3.3 Case C: Defect Shape

Case C is designed to study the relationship between defect shape and vibration response, with a rectangle defect defined for validation. The defect is located at 270°, the width and length are defined as 1.5×10^{-4} m and 3×10^{-4} m. With shaft frequency set as 1 Hz and radial load as -30 kN, the vibration signal is presented in Figure 11.

There are three peaks in a cycle, which appears at 4.5313 s, 4.6712 s and 4.8103 s respectively, as shown in Figure 11. These 3 peaks represent the time points when a ball enters and leaves the load zone, and then enters into the load zone again, respectively. Only the balls in load zone generate deformations, so the acceleration changes suddenly at the entry and exit of load zone. Therefore, acceleration between 4.5313 s and 4.6712 s in Figure 11 is the signal that occurs in defect zone.

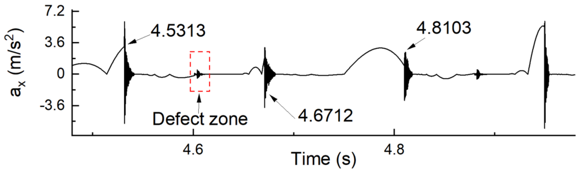


Figure 11. Acceleration in time-domain of bearing with a rectangle-shape defect.

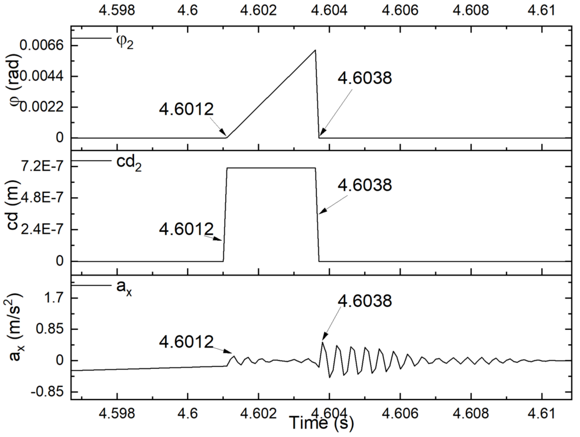


Figure 12. Signal output with a rectangle-shape defect: ϕ_2: angle between ball and defect start edge; cd_2: additional deformation of ball caused by defect; a_x: acceleration in x-direction.

To further study the signal in defect zone, the angle between ball center and the defect starting edge (ϕ_2), and the additional deformation generated by the defect (cd_2) are presented to demonstrate the transient process when the 2nd ball passes through the defect zone. As shown in Figure 12, ϕ_2 and cd_2 increase at 4.6012 s, indicating that the ball enters into the defect zone at this time. Thus, a_x shows an impulse at this moment. Likewise, the peak at 4.6038 s is the result of ball exiting because ϕ_2 changes to 0 at this point. The change of cd_2 presents a rectangle profile, which agrees well with the defined defect shape.

4 Conclusion

In this paper, a model of the whole bearing test bench including test bearing, connecting shaft, driving system and loading system is developed in OpenModelica. The proposed virtual test bench can be used to simulate bearing dynamics response, especially under different defect scenarios characterized by defect position, multiple defects, defect shape and defect size. It can be also employed as an alternative to a real test bench to generate fault signals for fault diagnosis algorithm development and validation, which could be a good supplement of experimental measurement when a large amount of data is required in machine learning or deep learning methods. The modeling theory and implementation process of the whole best bench are detailed, and three cases are designed to validate its effectiveness.

Due to the advantages in characteristics of open source, the OpenModelica has much superiority over the MATLAB/Simulink, furthermore, it also has more user-friendly interfaces with Python. In the future, the virtual bearing test bench developed in this paper will be adopted to study the transfer learning from the physics model to the real test bench.

Acknowledgements

This work is supported by CSC doctoral scholarship 201806250024 and Zhejiang Lab's International Talent Fund for Young Professionals.

References

Cong, Feiyun et al. (2013). "Vibration model of rolling element bearings in a rotor-bearing system for fault diagnosis". In: *Journal of sound and vibration* 332.8, pp. 2081–2097.

Cui, Lingli, Xue Chen, and Shujun Chen (2015). "Dynamics modeling and analysis of local fault of rolling element bearing". In: *Advances in Mechanical Engineering* 7.1, p. 262351.

Ho, D and RB Randall (2000). "Optimisation of bearing diagnostic techniques using simulated and actual bearing fault signals". In: *Mechanical systems and signal processing* 14.5, pp. 763–788.

Liu, Jing, Yimin Shao, and Teik C Lim (2012). "Vibration analysis of ball bearings with a localized defect applying piecewise response function". In: *Mechanism and Machine Theory* 56, pp. 156–169.

McFadden, PD and JD Smith (1984). "Model for the vibration produced by a single point defect in a rolling element bearing". In: *Journal of sound and vibration* 96.1, pp. 69–82.

Mishra, C, AK Samantaray, and G Chakraborty (2017). "Ball bearing defect models: A study of simulated and experimental fault signatures". In: *Journal of Sound and Vibration* 400, pp. 86–112.

Mitianiec, Wladyslaw and Jarosław Bac (2011). "Mathematical model of the hydraulic valve timing system". In: *Journal of KONES* 18, pp. 311–321.

Patel, VN, N Tandon, and RK Pandey (2014). "Vibrations generated by rolling element bearings having multiple local defects on races". In: *Procedia Technology* 14, pp. 312–319.

Randall, Robert B and Jerome Antoni (2011). "Rolling element bearing diagnostics—A tutorial". In: *Mechanical systems and signal processing* 25.2, pp. 485–520.

Rydberg, Karl-Erik (2016). *Hydraulic Servo Systems: Dynamic Properties and Control.*

Modelica Models as Integral Part of the Building Design Process

Dipl.-Ing. Torsten Schwan[1] Dipl.-Ing. Monika Wicke[1]
Dipl.-Ing. Alexander Hentschel[1] Dipl.-Ing. René Unger[1]

[1]EA Systems Dresden GmbH, Germany, {torsten.schwan, monika.wicke, rene.unger, alexander.hentschel}@ea-energie.de

Abstract

The design process of buildings and energy supply systems consists of several steps with increasing accuracy and decreasing fault tolerance. Because of a wide range of unknowns and increasing complexity, Modelica models are often an integral part of the first design steps. However, there are only rare updates and reuse of these models in later phases and/or during building use.

This paper emphasizes the potential of a continuous update and use of available Modelica models during all steps of building design processes. It therefore regards an example of a research greenhouse building for which the initially developed Modelica models were continuously updated and reused during the final phase of intensive scientific monitoring. Furthermore, general insights in latest scientific approaches indicate suitable steps of partly-automated continuous model updates during the whole building life span using BIM (Building Information Modeling).

Keywords: Building Simulation, Monitoring, BIM Integration, Integral Design Processes

1 Introduction

The building sector already represents one of the main fields of application of numeric simulation models. Especially, Modelica models therefore provide easy-to-use interfaces, a lively user community and a great variety of suitable toolsets and libraries. With its interdisciplinary physical modeling approach, Modelica enables engineers in many planning sections to analyze the cross-section behavior and influences of different system components, especially in the energy system with its HVAC and power supply units and a complex building control. Here, Modelica exploits its advantages in fast and accurate modeling of complex non-linear differential algebraic equation systems which are in this case caused by cross-domain system dependencies, volatile renewables availability and state-dependent storage behavior.

The design process of both newly constructed and retrofitted buildings follows an extensive, hierarchically-structured planning and implementation procedure which consists of three general phases and nine steps.

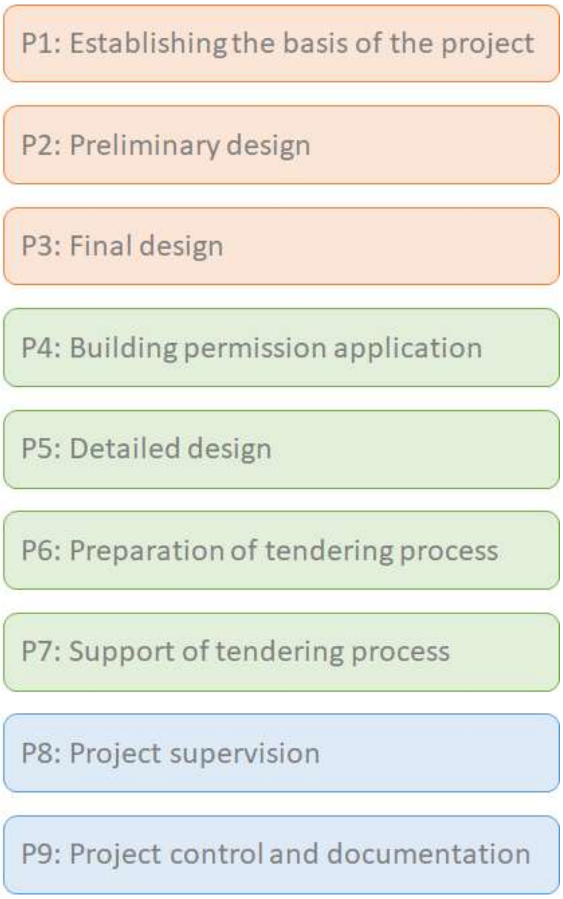

P1: Establishing the basis of the project
P2: Preliminary design
P3: Final design
P4: Building permission application
P5: Detailed design
P6: Preparation of tendering process
P7: Support of tendering process
P8: Project supervision
P9: Project control and documentation

Figure 1: Steps of German building design process (Sommer, 2016)

It starts with a brief design description and variant analysis (c.f. P1 & P2 in Figure 1) which were followed by the development of all relevant planning documents and the permission process (c.f. P3 in Figure 1).

A second phase (c.f. P4 to P7 in Figure 1) adds detailed evaluations and design plans which are then the base of the final construction (c.f. P8 & P9 in Figure 1). In case of complex designs and/or new design approaches, a certified engineering entity (i.e. university, engineering office) uses the available measurement data in a final two to three year monitoring phase to evaluate resulting system efficiency and possible optimization measures. This monitoring phase is basically part of P9 but actually represents a separate design process step.

If engineers decide to use Modelica models during this design process, their implementation normally starts in phase P2 to get initial design feedbacks. These models are then continuously updated during phase P3 regarding the increasing level of detail and required accuracy of results. However, further updates are not usual. The models remain at the P3 level and will mostly no longer be a part of later evaluations.

This is obviously not efficient as these models include a high amount of the engineering knowledge which partly gets lost when they won't be refined and reused. A better design process would take the models as basis of design knowledge (c.f. BIM – Building Information Modeling) and would align them as an integral part of the overall planning process. Therefore, they can be the device-under-test (i.e. DuT) of the final building controller evaluation or something comparable describing the reference behavior during the final monitoring phase.

2 Building Example

Building and energy system models based on the Modelica modeling language and derived libraries are mostly necessary to solve complex design decisions, like storages and renewables dimensioning or optimal control strategies. An exemplary building design process which addresses all of these challenges regarding the construction of a new research greenhouse building in the city center of Leipzig, a major city in East Germany. As a center of biological research, scientists and students of the University of Leipzig will use it to identify and evaluate effects of global warming on indigenous vegetation, and to perform further research relevant experiments (e.g. Craven et. al. 2019).

The first planning phases began in 2014 with some basic discussions of the main goals as well as preliminary design developments, c.f. 3D sketch of later building within the surrounding public park (GEFOMA, 2014). The building owner decided in the early stages to highlight high energy efficiency and low carbon footprint as major design goals besides versatile research equipment and restrained park integration.

During the first design steps, several solutions were discussed to cope with these challenges. Greenhouse buildings are commercial buildings with significant requirements on cooling power. The cooling system design was therefore recognized as important at an early design stage. Variant analysis showed that a partly solar-powered cooling system might have the best chance to meet the challenging goal of +50% carbon dioxide emissions reduction. Existing funding regulations required a mathematical verification of these potentials.

Because of the system complexity and the non-standard greenhouse building type, suitable models became necessary. They should describe both the cross-linked interaction of building and planting as well as the energy supply system partly including renewable cooling, heating and power supply. Furthermore, these models had to provide a sufficiently accurate comparison of the planned energy efficient design approach and a comparable reference solution.

A number of different greenhouse building simulation platforms and solutions were available (e.g. Rodríguez et. al., 2002). Even specific Modelica libraries have been developed since then (e.g. Altes-Buch et. al., 2019). However, Modelica models based on SimulationX and the Green City library were chosen to handle these challenges. The customized models developed - including a brief discussion of results - were already described in Schwan et. al, 2015.

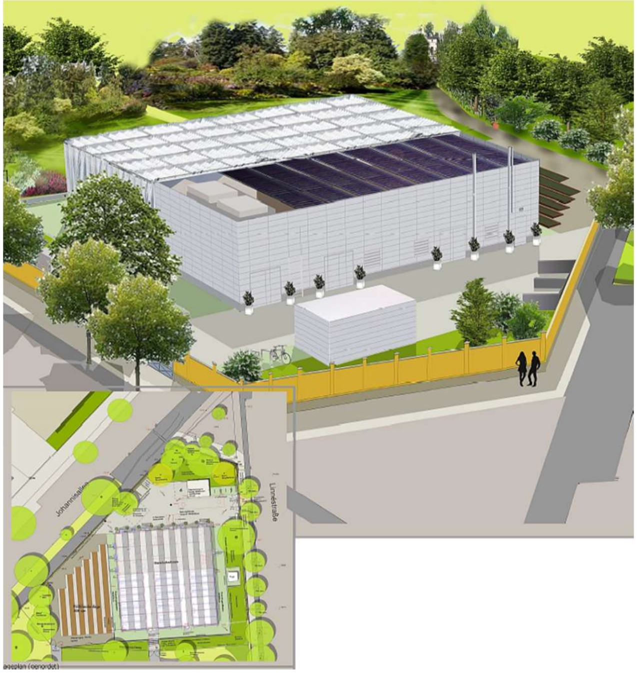


Figure 2: 3D digital mockup of the greenhouse building (GEFOMA, 2014)

All considerations and evaluations with the models of 2015 considered just the first design phase - including steps 1 to 3 in Figure 1. In a usual model-aided design process, the developed models would not be used or updated within the following design phases or even during the following building lifespan. However, this design process was different as the planned building included complex requirements on building use as well as sophisticated solutions of energy supply and building construction.

Nevertheless, the existing funding regulations demanded a long-term monitoring phase at the beginning of the use of the building. This monitoring phase includes both the evaluation of high-resolution measurement data of resulting system efficiency, as well as the proposal of

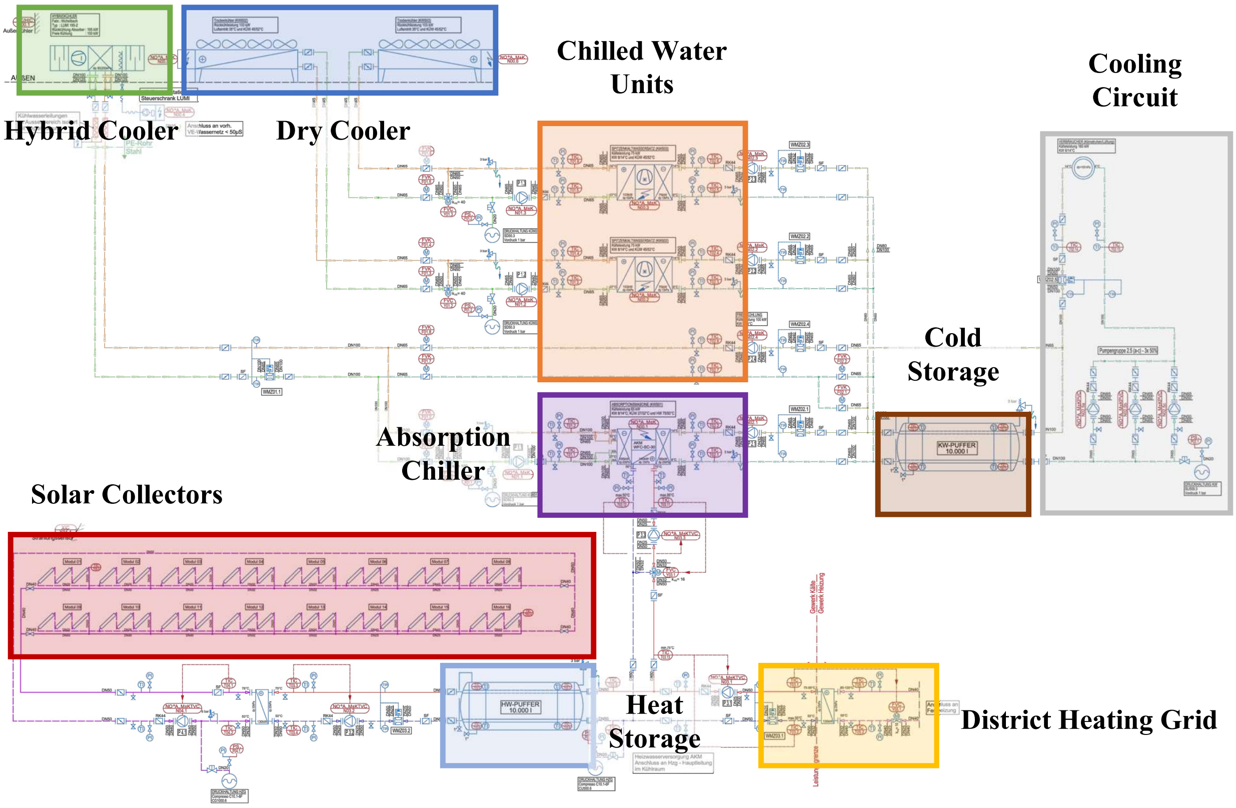


Figure 3: Cooling system concept at design step P2 (GEFOMA, 2014)

optimization measures. For both, the developed models are still necessary. On the one hand, the reference building model provides a source of comparison to evaluate the resulting carbon footprint savings. On the other hand, an updated model which is calibrated and enriched with available measurement data can provide further insight regarding alternative and optimized system solutions and control strategies. This is especially necessary if the monitoring indicates optimization potential which needs to be evaluated in detail regarding several suitable solutions prior to final real-world tests.

3 Continuous Model Refinement

To become an integral part of the building design process, Modelica models need to be refined continuously during the whole process as well as during the whole building life span in case of significant changes. This work effort considers on the one hand the model updates regarding increasing release versions of Modelica language and the simulation environment. The greenhouse building design process started in 2014 and ended in 2020 which included an update for Modelica from version 3.3 to 3.4 as well as several updates of SimulationX from version 3.6 to 4.1.

On the other hand, there are several evaluation steps of assumptions, accuracy tolerances and levels of detail between the different phases and design process steps. This often causes significant changes of the initially developed models depending on the design progress as well as the feedback to model requirements.

The first phase of the considered greenhouse building design process required Modelica models of both the energy supply system and the building including the crops. This was necessary to evaluate both savings potential from an improved building envelope and a better shading system as well as an increased environmental energy use via solar cooling. In later design phases, especially during the monitoring phase, the implemented cooling system model became more important. Monitoring data was then used to calibrate the model components and control, and to represent the building loads of cooling power consumption.

This paper focuses on the development steps and use of the cooling system model and neglects any simultaneous progress of building or heating system models. Therefore, Figure 3 shows the initial cooling system concept at design step P2 (c.f. Figure 1).

This concept described a bivalent cooling power supply by an absorption chiller and two peak-power chilled water units. The thermal compressor used solar heat from two types of solar collectors and heat from the local available district heating grid as a heat source to provide basic cooling power. A hybrid cooler is used as the recooler, which always ensures a recooling temperature of less than 27°C. It furthermore produces additional cooling power via free cooling in times of cold outdoor temperatures.

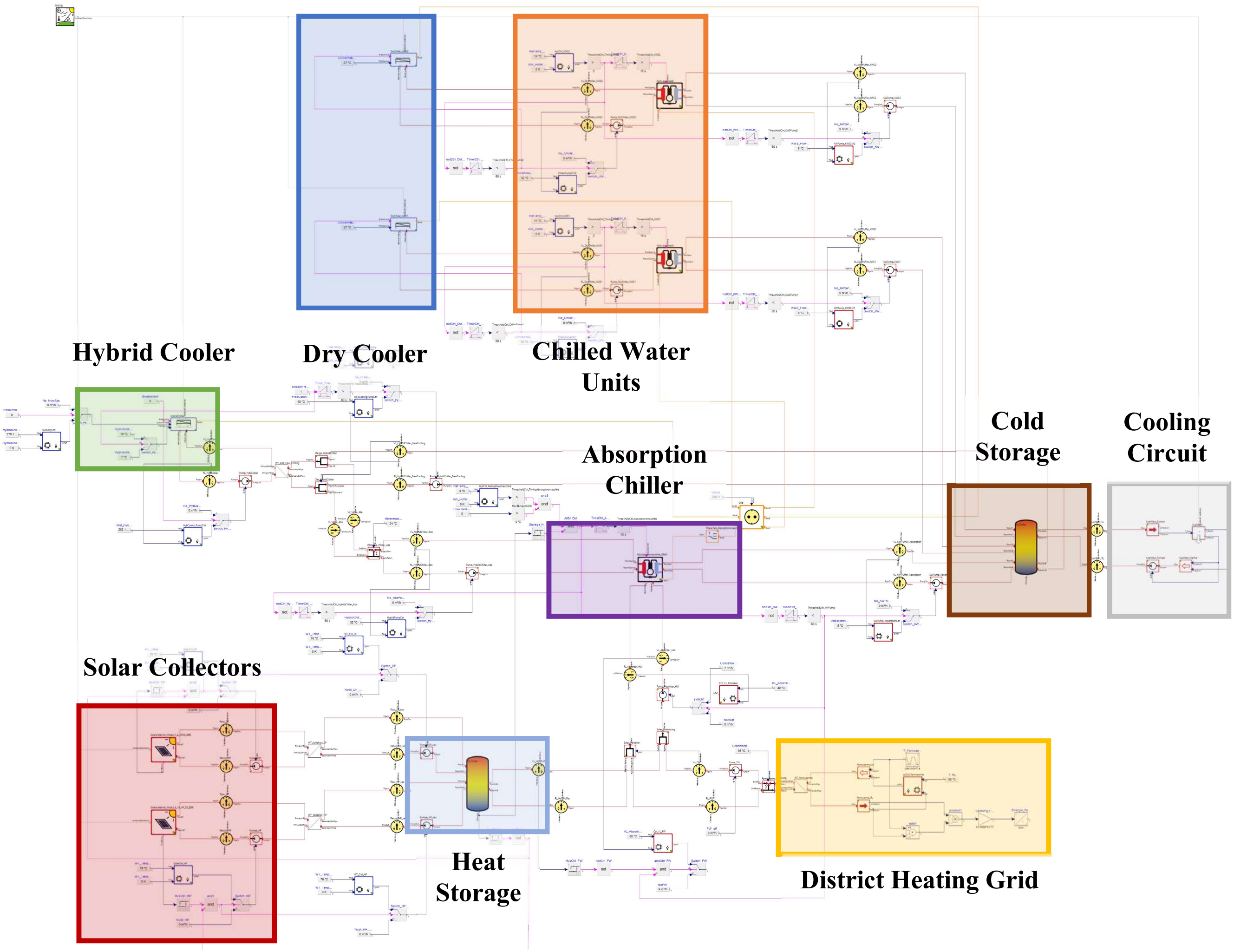


Figure 4: Simulation model at design step P2 (Schwan et. al., 2015)

This provides additional savings potential because greenhouse buildings continuously require high light intensity and thus cause significant cooling load even during winter and transit time periods.

Cooling power peaks should be buffered with the 10 m^3 cooling water tank and the chilled water units. Both tank systems were initially planned as single lying tanks underneath the laboratories. The chilled water units only use dry coolers as recoolers because of reduced requirements on recooling temperatures. These dry coolers use a water-glycol mixture as heating medium to avoid freezing outside the building shell. In contrast, the hybrid cooler was planned to use water with an electric trace heating system to avoid temperature drops at additional heat exchangers for free cooling.

All these constraints were then used to model an adequate mathematical and physical representation using the Modelica language and derived simulation libraries. Figure 4 shows this model which was based on the former Green Building library in SimulationX (c.f. Schwan et. al., 2015).

It almost shows a one-to-one representation of all relevant system and control components as well as their hydraulic and electric configuration and connections. The individual components, such as absorption chiller and chilled water units, used system parameters and operating characteristics which were derived from data sheets of typical system manufacturers but not from measurements. The simulated total system efficiency thus significantly depended on accuracy of this data, especially the EER characteristic of the chilled water units and the COP of the absorption chiller. The final P2 model evaluations showed a total carbon footprint saving potential for the cooling demands of about 51.06%, which is only about 1% higher than the design goal of 50%. This potential included both the savings of solar cooling and a better energetic standard of the building envelope, and a smart shading system.

The overall design process including final tests of building, planting area and energy supply system took over 6 years. Since the beginning, there have been an ongoing process of design updates with respect to system details and accuracy. The P2 model was always refined in

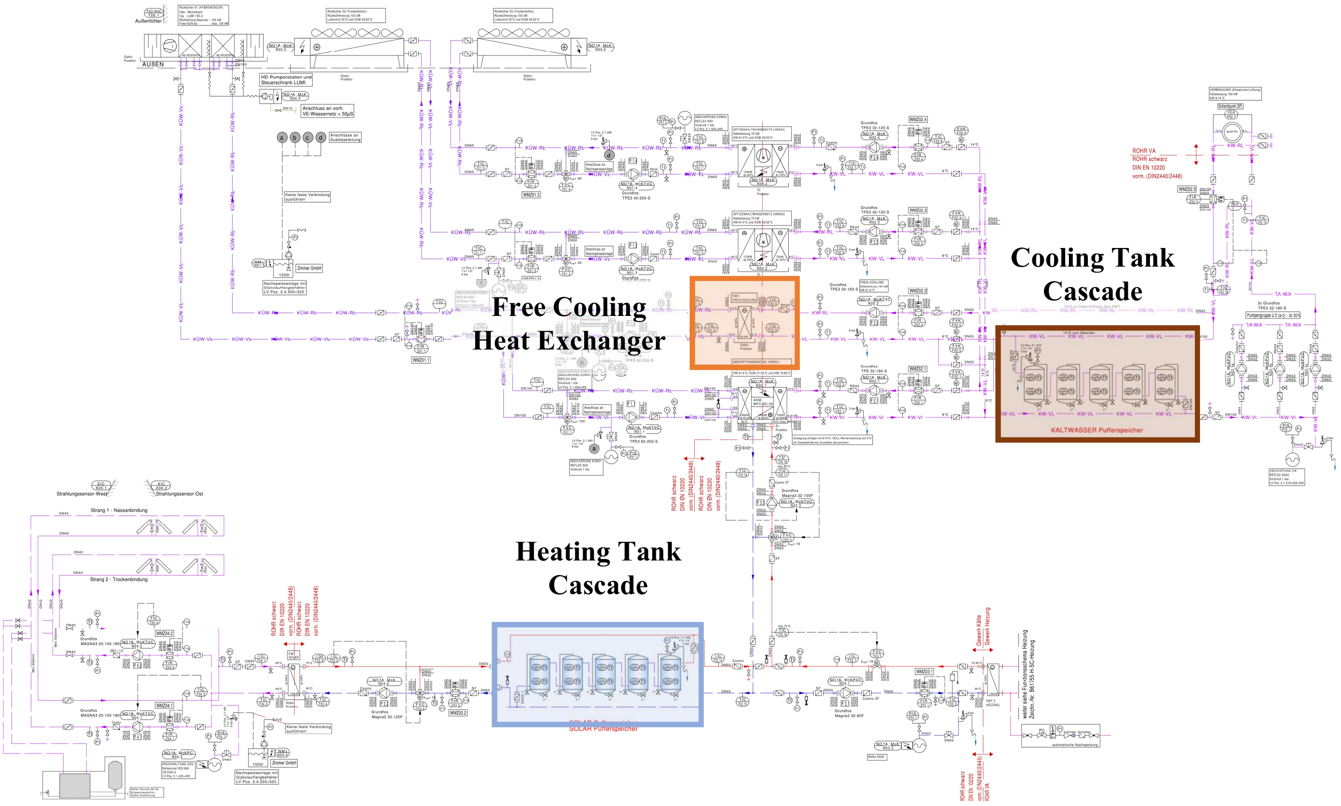


Figure 5: Cooling system concept at design step P9 (Zimmer, 2017)

order to be as up to date as possible and to be available for any design question.

Already in P3, the final design step, the building owner decided to change the hydraulic integration as well as the heating medium of the hybrid chiller for system safety reasons. An additional heat exchanger was therefore necessary which however caused an additional temperature drop and thus additional reductions of free cooling potential. Analyses of the updated model showed that this only slightly reduced the total carbon footprint saving potential to about 50.08%, still above the major design goal level.

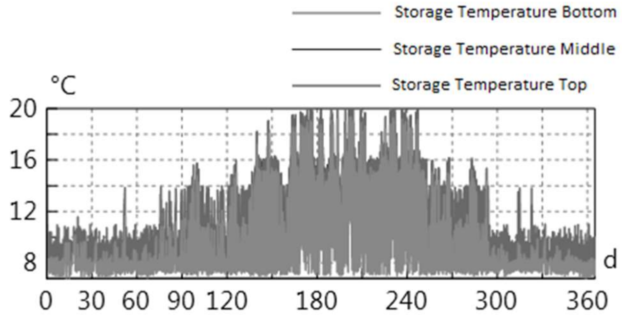


Figure 6: Simulated cold storage temperature with the P2 model (Schwan et. al., 2015)

Further evaluations during the design process at the P5 step showed that the lying heat and cooling storage tanks forced the mixture of the heating medium inside. Because of their configuration, they would be responsible for very low temperature spreads in both the cooling system as well as the heat supply circuit of the absorption chiller. This resulted in a significant increase of the required circulation pumps volume flow and thus necessary costs and auxiliary power consumption.

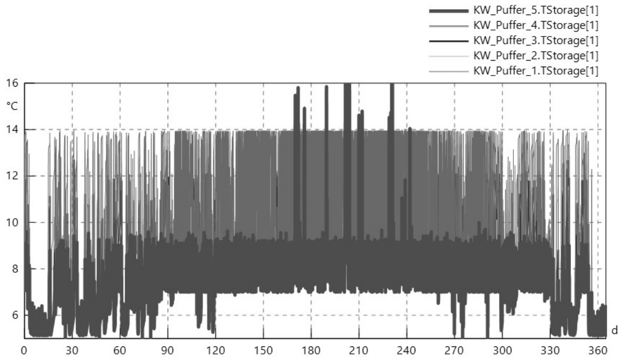


Figure 7: Simulated storage tank temperatures with updated P5 model

To avoid these short circuits, both storage systems have been changed to tank cascades with subsequent storage tanks of 1/5 of the original storage size, each within design step P5. With hydraulic connections between the top of the previous and the bottom of the next tank, the temperature difference between storage system input and output could be increased to a maximum level.

These design decisions could be technically supported

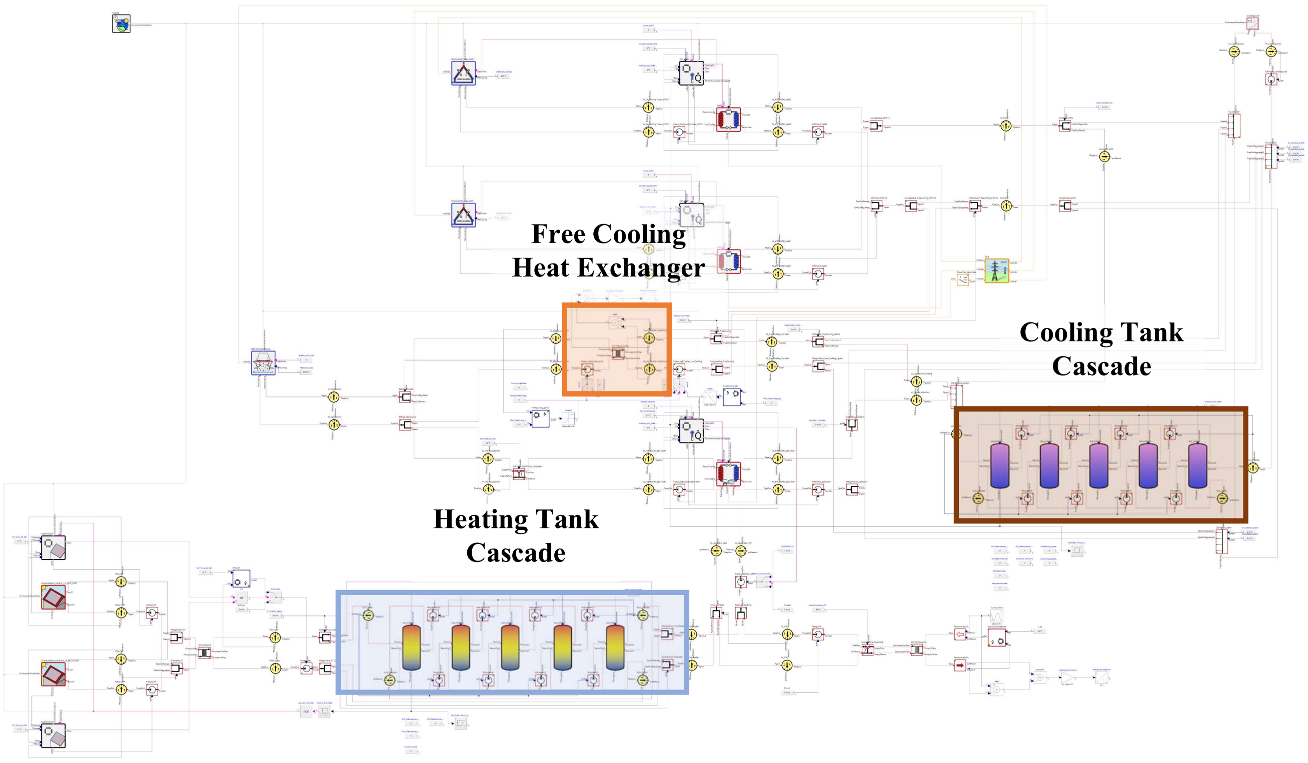


Figure 8: Simulation model at design step P9

and validated by the adapted simulation models. The results in Figure 7 in comparison to the former system behavior evaluated in Figure 6 show the effectiveness of these measures concerning the increase of storage temperature spread and cooling system temperature reduction.

Further model updates during the design progress mainly considered the level of detail of the implemented control strategies as well as further insights from the increasing availability of measurement data obtained from monitoring and surrounding conditions.

The original control strategy defined the absorption chiller as the basic cooling power source. If the solar heat in the storage tank(s) was too low, remaining heat was planned to be taken from the local district heating grid. The implemented Modelica models showed that during P2/P3 evaluations the solar collectors would provide only about 30% of the required heat demand for cooling. However, this was still efficient because of the expected performance characteristic of the chosen equipment.

First tests after the implementation showed significant influence of the heating system temperatures and temperature spreads on the total absorption chiller efficiency. This became another one of the unexpected issues, because the desired system efficiency required significantly higher heating system temperatures in both flow (85°C instead of 75°C) and return (80°C instead of 50°C). However, the return temperature of the district heating grid is limited to 55°C on the primary side.

|Thus, the originally planned system control is not possible. The cooling system simulation model was therefore updated according to latest analysis of system operation and control as well as measured system parameters. Further analyses compared alternative solutions which were necessary to still achieve the major design goal of 50% plus carbon footprint saving potential. Therefore, available measurement data from the monitoring as well as latest design documents were used to again update the model regarding the final P9 system status. This included updates of model components, parameters, and hydraulic connections and nevertheless integrated control algorithms (e.g. absorption chiller start-up procedure depending on heat tank temperatures). Furthermore, measurement time series of the total cooling power consumption partly replaced the previous simulated load curves.

4 Design Questions to the Models

One of the final measures in the P9 design step was the final adjustment of the system control after the construction of the building and energy system is finished. Especially, the absorption chiller operation in particular showed significant gaps between design phase and final implementation.

The main issue was the implemented heat supply circuit of the absorption chiller. It was designed to enable 50°C maximum return temperature with a 25K temperature spread because of district heating grid requirements. However, this caused significantly lower system efficiency as the absorption chiller required a lower temperature spread of about 5K at a higher temperature level to provide adequate efficiency ratios. Monitoring data of measured heat supply and recooling temperature as well as resulting cooling power were used to calibrate the corresponding model characteristics. Figure *9* shows some exemplary results of the calibration process.

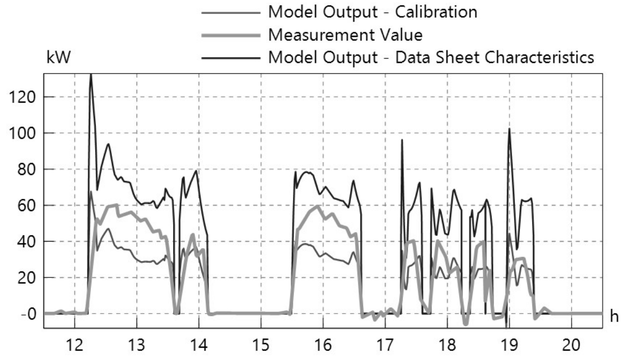


Figure 9: Comparison of measurement data and model results during the calibration process

Again, a carbon footprint saving potential of 50% of a comparable reference greenhouse building remained the major design goal. For this purpose, the engineers developed four technically-feasible solutions which might help to compensate the resulting efficiency reduction due to the changed boundary conditions.

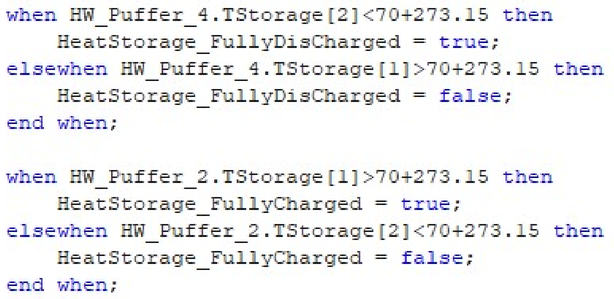
1. Replacement of absorption chiller with a machine with better efficiency ratio at the desired temperature level (i.e. 75°C)
2. Variable control of cooling power output depending on available solar heat storage tank temperatures (i.e. 60°C to 75°C)
3. Increase of heat supply temperature level (i.e. 85°C)
4. Bivalent heat supply from solar heat storage tanks and district heating grid

The first solution required significantly higher investment costs than the other ones because it considered the replacement of an already integrated system component- the complete absorption chiller including all peripheral components. In contrast, the second and third option would not need additional investments besides the engineering effort regarding the required controller adaption.

The last solution only represented an optional way to show the entire range of the technically-feasible approaches. However, it would violate the requirements of district heating gird because maximum return temperatures of 55°C would not be allowed.

To find the right solution regarding the major design goal, the continuously updated Modelica model of the cooling system is predetermined. It is precisely here that the strength of Modelica's approach of making models usable for the entire design period up to the building's use becomes apparent.

The required Modelica models must therefore represent the real-world conditions as accurately as possible to support those important design decisions. Intensive calibration work and structural redesign of the hydraulic model at the end of design phase P9 ensured this accuracy regarding the physical system behavior. However, effects of the control system are almost as important as the component parameters and model structure. Therefore, the complete technical description of the control system was implemented using all available terms of the Modelica language.



```
when HW_Puffer_4.TStorage[2]<70+273.15 then
    HeatStorage_FullyDisCharged = true;
elsewhen HW_Puffer_4.TStorage[1]>70+273.15 then
    HeatStorage_FullyDisCharged = false;
end when;

when HW_Puffer_2.TStorage[1]>70+273.15 then
    HeatStorage_FullyCharged = true;
elsewhen HW_Puffer_2.TStorage[2]<70+273.15 then
    HeatStorage_FullyCharged = false;
end when;
```

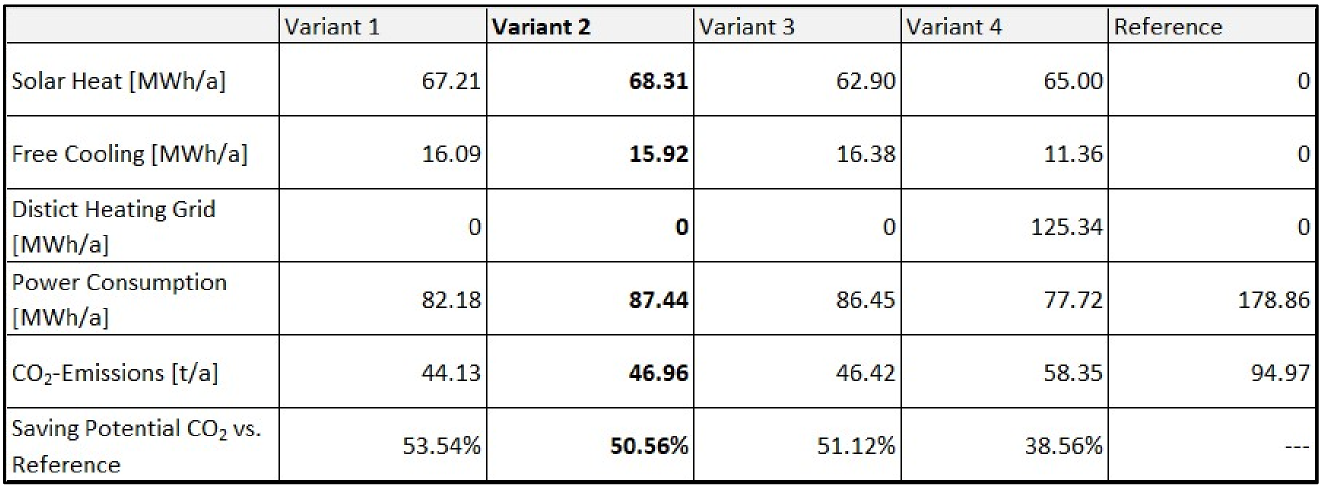
Figure 10: Section of the Modelica controller code of the storage temperature control

Figure *10* therefore shows a small exemplary section of the implemented controller code. It decides if the solar heat storage reached the level “fully-discharged” or “fully-charged” depending on simulated temperatures in different tanks and in different tank positions. This again shows the strength of Modelica models regarding the support of the whole building design process. Specific problems or even a complex system can be modeled in different levels of representation (i.e. structural and text view) and detail.

The final evaluation of the four simulation model variants provides a conclusive result. Table *1* therefore shows some decision-making factors. The calculation of the total CO_2 emissions per year requires the evaluation of all energy flows, especially from fossil fuels or grid power, to the building. Therefore, power consumption from the electric grid and heat consumption from the district heating grid are the most important values. They will be multiplied with their individual CO_2 emission equivalent factors (i.e. power: 0.54kg/kWh, district heating: 0.15kg/kWh) to calculate the simulated total CO_2 emissions of each variant.

Furthermore, Table *1* also shows the generated local renewable energy amounts. This includes renewable cooling power via free cooling with the hybrid cooler and solar heat from the two types of solar collectors. As renewables, they are not part of the CO_2 emissions

Table 1: Simulation results of decision making factors regarding the analysis of variants



	Variant 1	**Variant 2**	Variant 3	Variant 4	Reference
Solar Heat [MWh/a]	67.21	**68.31**	62.90	65.00	0
Free Cooling [MWh/a]	16.09	**15.92**	16.38	11.36	0
Distict Heating Grid [MWh/a]	0	**0**	0	125.34	0
Power Consumption [MWh/a]	82.18	**87.44**	86.45	77.72	178.86
CO$_2$-Emissions [t/a]	44.13	**46.96**	46.42	58.35	94.97
Saving Potential CO$_2$ vs. Reference	53.54%	**50.56%**	51.12%	38.56%	---

calculation (i.e. CO$_2$ factor is 0kg/kWh) but they are listed in Table *1* as well to enable a detailed discussion of different influencing factors.

The last column of Table *1* includes the corresponding values of a fictional reference greenhouse building with a lower energetic building standard and conventional cooling system with three independent chilled water units. The calculated CO$_2$ emissions of each variant are compared to this reference solution to evaluate the savings potential.

In contrast to the results of the first design phase, a bivalent heat supply to the absorption chiller via solar heat collectors and district heating grid (i.e. variant 4) doesn't meet the major design goal anymore. The total CO$_2$ savings potential decrease below 40%. The measurements showed that the deciding temperature level of the absorption chiller results from average temperature between flow and return (i.e. 75/50°C => 62.5°C) which is much lower than the estimated temperature level during the design phases P2/P3 (i.e. 75°C). The lower the heat supply temperatures, the lower the absorption chiller performance which results significantly higher district heating consumption and higher CO$_2$ emissions. Furthermore, this control variant also significantly reduces the free cooling potential because the cooling tank temperature mostly remains on a low level because of the infinite availability of district heating.

Variants 1 to 3 all meet the major design goal of 50% CO$_2$ saving potential. The higher heat supply temperature level (i.e. 85°C) of variant 3 results in a higher cooling power output of the absorption chiller which reduces the operation time of the peak-power units. However, this higher temperature level causes a lower solar collector efficiency which is compensated by the better performance characteristic of the absorption machine.

A new machine with a better performance characteristic (i.e. variant 1 – 75°C) enables both a higher cooling power consumption and higher solar collector efficiency. However, this variant requires significantly higher investment costs.

Variant 2 requires the lowest investment costs as it needs only minor changes of the control strategy, and as such it is preferred over variant 1. Variant 3 is more expensive as well because the system of internal volume flows (i.e. pumps and piping) must be converted to other dimensions. The presented use of Modelica models provided a conclusive design recommendation of variant 1 even at a very late step of the design process.

5 Process Integration

This paper presents the use of Modelica models as an integral part of the complete design process. However, complex commercial buildings in particular still require a high level of manual effort for model updating and process synchronization.

However, there are already ongoing processes and scientific research, i.e. German FMI4BIM project, which works on solutions which partly link simulation models and derivatives (i.e. FMUs) to typical digital data platforms used within building design processes (i.e. BIM – Building Information Modeling). This approach already shows a lot of application scenarios (c.f. Eckstädt et. al., 2020).

Building Information Modeling (BIM) is a core concept of Industry 4.0 mainly used by the construction industry as a consistent approach of data management during the whole design and construction process (c.f. Doan et. al., 2019). It was once designed to provide a database of building construction data, parameters and configurations but is now extending to additional engineering domains of the building sector, especially HVAC systems and building control.

Therefore, BIM is predestinated to serve as an

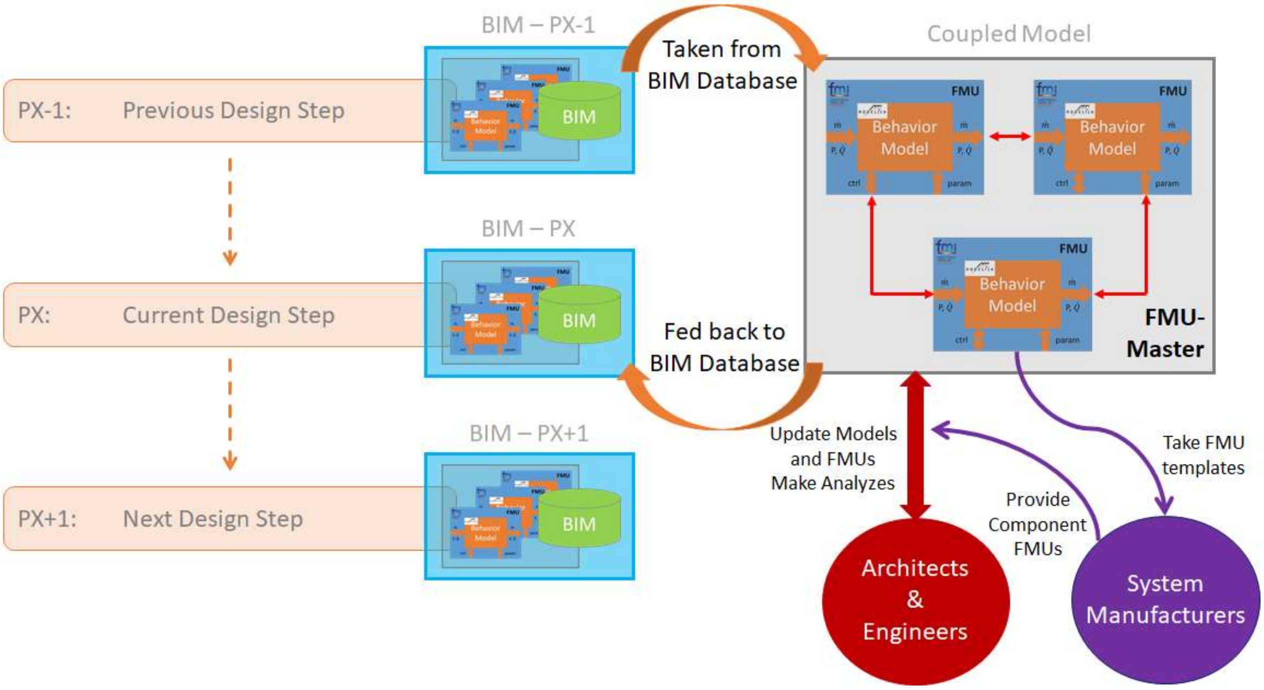


Figure 11: Basic approach of Modelica model (FMU) integration in BIM-based building design process

independent model database during the whole building design process and the following building lifespan. To increase interoperability and tool-independency, Modelica models should therefore be converted to FMUs as it is proposed by the FMI4BIM consortium.

Therefore, BIM is predestinated to serve as an independent model database during the whole building design process and the following building lifespan. To increase interoperability and tool-independency, Modelica models should therefore be converted to FMUs as it is proposed by the FMI4BIM consortium.

Figure *11* describes the basic approach of a BIM-based building design process. All architectural data is stored in the BIM database. Architects and engineers use these data with each design step and refine the collected data and information in the BIM model regarding the increasing design knowledge and accuracy. Additional links to suitable building and HVAC system FMUs (or coupled FMU models) can extend this process to enable a full integration of Modelica models.

The share of BIM-based building construction projects is constantly increasing worldwide (c.f. Liu et. al., 2021). Typical CAD tools began to integrate ifc-files (i.e. BIM file format) export and import. BIM has become an important issue in the field of building engineering. There are even toolchains that already exist that automatically generate and parameterize Modelica models with BIM data (c.f. Nytsch-Geusen et. al., 2019). All these tools and methods contribute to a more consistent building design process and facility management.

Basically, the approach in Figure *11* extends the existing BIM-based process with suitable links to models represented by FMUs or FMUs libraries. Therefore, the assigned model environment is updated with new parameters and information during each design step. Then, engineers and architects use the models for individual analysis, e.g. energetic evaluation of different HVAC system variants. The most preferred solution is then fed back to the BIM environment as base for the next design steps.

The model design, interfaces and FMU integration will be defined by a standard or standard extension. This will allow system manufacturers to provide individual component FMUs of their products based on the developed standard templates. This will contribute to both security of intellectual property as well as integration of expert knowledge.

6 Conclusion

The presented example building design process of the new research greenhouse building of the University of Leipzig shows the versatility of Modelica models regarding upcoming design questions. Currently, Modelica models are often used as base of decision-making in the first design phase until step P3, the final design. However, there is rarely any use of these models after this phase.

The different design steps and phases have different requirements regarding accuracy and level of detail. The use of these models in subsequent design phases needs continuous refinement and a consistent data base.

Therefore, BIM seems to be the means of choice as it already represents a digital mockup of the continuously refined building construction during all design phases. Ongoing research activities focus on the integration of different toolsets and databases, like BIM, GIS and Modelica models. Open-source frameworks are thus available to use BIM data in Modelica models (c.f. Wetter, et.al., 2019).

Another research activity, i.e. FMI4BIM, currently analyses approaches of a full BIM process integration of Modelica models. Thus, a project specific BIM model should be linked to the Modelica models which are represented here by FMUs to provide a tool-independent model exchange standard. One major outcome of the presented exemplary greenhouse building construction process considers the total process length corresponding to the desired BIM workflow. Design and construction of complex buildings can easily take several years. If Modelica models (or derived FMUs) should be linked to different process steps, a well-suited version management is necessary. This doesn't only consider the model's level of detail and accuracy, but also the version of current Modelica language and Functional Mockup Interface release, and the used simulation environment which is also important. This is a particular challenge when backward compatibility needs to be ensured.

Models can help identify optimal solutions of specific design questions during all design steps via variant analysis and parameter study. In case of a continuous model refinement process, models can also serve as virtual devices-under-test for the building control development. This requires a significantly automated process including controller design until the phase where controller code is exported to specific targets (i.e. PLCs, DDCs, etc.). Further research activities, such as ARCHE, have special emphasis on this topic. This includes the consistent decoupling of controller code and physical model. Therefore, the Modelica Synchronous library provides suitable approaches which describe boundaries between clocked and continuous-time partitions of a model (c.f. Elmquist et. al., 2012).

7 Acknowledgements

Supported by:

Federal Ministry
for Economic Affairs
and Energy

on the basis of a decision
by the German Bundestag

The presented research work and results were partly encouraged by subsidies of the German Federal Ministry of Economics and Technology.

References

Sommer, H., Projektmanagement im Hochbau - Mit BIM und LEAN Management, Heidelberg, Berlin: Springer-Verlag, 2016.

Craven, Dylan, Winter, Marten, Hotzel, Konstantin, Gaikwad, Jitendra, Eisenhauer, Nico, Hohmuth, Martin, König-Ries, Birgitta, Wirth, Christian (2019). Evolution of interdisciplinarity in biodiversity science. In: *Ecology and Evolution* 9(12), 6744-6755.

GEFOMA Großbeeren GmbH. IDIV greenhouse building construction in Leipzig. Design documents, 2014.

Rodríguez, F., Yebra, L.J., Berenguel, M., Dormido, S. (2002). Modelling and Simulation of Greenhouse Climate using Dymola, In: *IFAC Proceedings Volumes, Volume 35, Issue 1, 2002, Pages 79-84*.

Altes-Buch, Queralt & Quoilin, Sylvain & Lemort, Vincent. (2019). Greenhouses: A Modelica Library for the Simulation of Greenhouse Climate and Energy Systems. In: *533-542. 10.3384/ecp19157533*.

Schwan, Torsten, Unger, René, Pipiorke, Jörg (2015). Energy-Efficient Design of a Research Greenhouse with Modelica. In: *11th International Modelica Conference*. Versailles, 2015.

Zimmer & Hälbig GmbH. IDIV greenhouse construction in Leipzig. Revision documents, 2017.

Eckstädt, Elisabeth, Paepcke, Anne, Hentschel, Alexander, Schneider, André, Nicolai, Andreas, Schumann, Falk (2020). Simulationsszenarien für Gebäude-Energiesimulation in frühen Planungsphasen. In: *BauSIM 2020*. TU Graz, 2020.

Doan, Dat & Ghaffarianhoseini, Ali & Naismith, Nicola & Zhang, Tongrui & Rehman, Attiq Ur & Tookey, John & Ghaffarianhoseini, Amirhosein. (2019). What is BIM? A Need for a Unique BIM Definition. In: *MATEC Web of Conferences. 266. 05005. 10.1051/matecconf/201926605005*.

Liu, Ziwen Lu, Yujie, Shen, Meng, Peh, Lu Chang (2021). Transition from building information modeling (BIM) to integrated digital delivery (IDD) in sustainable building management: A knowledge discovery approach based review. In: *Journal of cleaner production 2021 v.291 pp. 125223*.

Nytsch-Geusen, Christoph, Rädler, Jörg, Thorade, Matthias, Ribas Tugores, Charles (2019). BIM2Modelica – An open source toolchain for generating and simulating thermal multi-zone building models by using structured data from BIM models. In: *13th International Modelica Conference*. Regensburg, 2019.

Wetter, M., von Treck, C., Helson, L., Maccarini, A., Saelens, D., Robinson, D., Schweiger, G. (2019). BIM/GIS and Modelica framework for building and community energy system design and operation – ongoing developments, lessons learned and challenges. In: *Sustainable Built Environment Conference*. Graz, 2019.

Elmquist, Hilding, Otter, Martin, Mattsson, Sven Erik (2012). Fundamentals of Synchronous Control in Modelica. In: *9th International Modelica Conference*. Munich, 2012.

A Cloud-native Implementation of the *Simulation as a Service*-Concept Based on FMI

Moritz Stüber[1] Georg Frey[1]

[1]Chair of Automation and Energy Systems, Saarland University, Germany,
{moritz.stueber,georg.frey}@aut.uni-saarland.de

Abstract

Providing modelling and simulation capabilities *as a service* promises to increase their value by improving accessibility for non-expert users and software agents as well as by leveraging cloud-computing technology to scale simulation performance beyond the capabilities of a single computer. In order to reach this potential, implementations must align their design with the architectural styles of cloud computing applications and the web in general. We present an open-source, cloud-native Simulation as a Service (SIMaaS)-implementation that gives access to models and allows simulating them on the web. The implementation uses Functional Mockup Units (FMUs) for co-simulation as an executable form of a model and relies on FMPy for simulation. It is realized as a microservice in the form of a REST-based HTTP-API. Functionality and performance are demonstrated by using the service to create ensemble forecasts for PV systems and to search for an optimal parameter set using a genetic algorithm. Conceptual limitations and the resulting opportunities for further work are summarized.

Keywords: simulation as a service, cloud-native simulation, service-oriented software architecture, FMI 2.0

1 Introduction

There exist scenarios in which it is useful to execute simulations on a distributed set of computing resources that can be scaled according to demand and beyond the capabilities of a single machine. Examples for this include simulations which are part of a series of many simulations to be evaluated as a whole; as for example in parameter fitting or sensitivity analysis applications. Also, simulations might be part of a (recurring) larger process, such as providing necessary forecasts for flexibility management in the context of smart grids.

The term *cloud computing* denotes a set of desirable characteristics for accessing a set of computing resources over the internet, as well as characteristic service models and deployment models (Mell and Grance 2011). From a user's point of view, the essential characteristics are that software or computing resources are available *as a service* via the internet, meaning that the resources are readily available without the need for manual installation of hardware and/or software. Consumers can use services without the need for human activity on the side of the provider (*on-demand self-service*), often without apparent limitations, and they have access to metrics for their service usage (*measured service*). Users are billed according to service usage in terms of these metrics (*pays-as-you-go cost model*).

Cloud-based end-user applications usually integrate several services to realize their functionality as it has been found that programmers can effectively realize the desirable characteristics of cloud computing by exposing pieces of functionality as a set of independent services in a so-called Service-oriented Architecture (SOA). In the abstract, SOA is "a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains" (OASIS 2006, line 864), where a *service* is defined as the "mechanism by which needs and capabilities are brought together" (OASIS 2006, line 174). More specifically, a service can be seen as the offer to perform work for others; as the service interface which specifies information model, behaviour model and applicable usage policies; and as a specific service instance.

In practice, SOAs can be successfully realized as a set of *microservices* in the form of Representational State Transfer (REST)-based Hypertext Transfer Protocol (HTTP)-Application Programming Interfaces (APIs) which exchange machine-readable representations such as JavaScript Object Notation (JSON) and define their interface according to a formal specification such as the OpenAPI Specification (OAS). The term microservice is used to point out that services best implement exactly one functionality only ("do one thing well"). Representational State Transfer (REST) is the name of the architectural style of the web, in other words a name for its key design principles. Consequently, a REST-based[1] HTTP-API attempts to implement these design principles, acknowledging the sucess of the web and attempting to inherit its positive properties.

Applications which are intentionally designed to work well in the cloud and consequently realize the desired characteristics are called Cloud-native Applications (CNAs). The goal of the presented work is to pro-

[1] Using the term REST-based instead of RESTful indicates that the developers are aware that the term "RESTful" is frequently misused and that their software does not fully realize the REST constraints.

vide a state of the art Simulation as a Service (SIMaaS)-implementation based on an established open standard for model export and to represent the ability to perform simulations as REST-compatible as possible withouth actually realizing the so-called Hypermedia As The Engine Of Application State (HATEOAS) constraint. Furthermore, it should be shown that the desired characteristics implied by the term "cloud-native" are realized.

The remainder of this paper is organized as follows: first, the concepts and abstractions for providing software *as a service* in general and the motivation for providing modelling and simulation as a service (MSaaS) are outlined in section 2. The high-level requirements that follow from the choice of concepts are summarized. Second, the software architecture and software stack for the developed solution are explained in section 3. Restrictions posed on Functional Mockup Units (FMUs) to be used with the implemented software are stated. Third, exemplary use cases for demonstrating functionality and performance are described in section 4. Last, related work is outlined in subsection 5.1 and conceptual limitations of the devised solutions as well as the resulting opportunities for further research are discussed in subsection 5.2.

2 Concepts

Providing MSaaS is a multi-faceted endeavour at the intersection of modelling and simulation (M&S), information science and software development and -operations (DevOps). The core hypothesis of MSaaS is that usability and reuse can be increased by making M&S functionality available to a broader audience via the internet; that functionality can be improved by facilitating the composition of M&S resources; and that performance can be improved by deploying applications in the cloud (see Stüber, Exel, and Frey 2018, section 2 for a detailed explanation and references to original research).

Three recent reviews on MSaaS outline the design space and identify high-level requirements and architectural choices that should guide the design and implementation of specific MSaaS solutions.

First, Shahin, Babar, and Chauhan map out the Architecture Design Space (ADS) for MSaaS by presenting the results of a Systematic Literature Review (SLR) performed with the aim to identify and describe the state of the art (Shahin, Babar, and Chauhan 2020). They categorize the primary studies considered according to different criteria such as the architectural style, the main drivers for architectural decisions, and quality attributes. Additionally, they ponder the implications of the chosen architectures and identify strenghts and weaknesses. The authors conclude that MSaaS-realizations most often use a *layered approach* to build applications; that *containerization* is employed to improve deployability; and that *effective interfaces for end users* that hide complexity and technicalities motivate their development (Shahin, Babar, and Chauhan 2020, section 5).

Second, Hannay, Berg, et al. (2020) reason about the infrastructure capabilities they deem necessary for realizing entire MSaaS ecosystems at scale. Based on "a systematization of concepts from ongoing deliberations on MSaaS" (Hannay, Berg, et al. 2020, section 3), the authors first review the service concepts of the North Atlantic Treaty Organization (NATO) MSaaS reference architecture (Hannay and Berg 2017) and then elaborate on the functionality required for realizing MSaaS ecosystems. Their reasoning is structured around the themes *data management*, *service description and -discovery*, *composition and interoperability* and the *management of different components*. The findings are mostly conceptual in nature, likely useful for verbalizing and contextualizing design questions and -decisions when faced with implementing specific SOAs containing M&S capabilities. The authors also note that solutions for supporting, yet – from an operational perspective – highly relevant functionality such as logging, metering and monitoring are readily available.

Third, Kratzke and Siegfried (2020) focus on the consequences expected from leveraging the cloud for M&S services. Using their work on and definition of CNAs (Kratzke and Quint 2017) as a basis, they propose a definition for what Cloud-native Simulations (CNSs) are in terms of a textual definition (Kratzke and Siegfried 2020, section 4.3), a cloud-native simulation stack and a cloud simulation maturity model. They summarize the software engineering trends in cloud computing as the evolution of deployment strategies to maximize resource utilization (smaller deployment units, elasticity); the use of microservices as an architectural style that supports the aforementioned; and the emergence of microservice engineering ecosystem components for container orchestration, monitoring, et cetera. The authors conclude that the same trends are to be expected for CNS architectures and that, like CNAs in general, CNSs should strive to isolate state in a minimum of stateful components.

In alignment with the findings of Kratzke and Siegfried, section 4.2, we decided to create a microservice realizing the SIMaaS-concept in the form of a REST-based HTTP-API, relying on containers as deployment units to be operated on a clustered elastic platform.

As a consequence of the decision for an interface design based on REST, specifically the *uniform interface constraints*, M&S capabilities need to be represented as *resources* of which *representations* can be transferred when HTTP verbs are applied to them (Verborgh, Hooland, et al. 2015). In other words, a mapping between entities of the application domain, such as a models and simulation results, and uniquely identifiable conceptual resources that constitute the service interface is required.

In the context of the developed SIMaaS-API, the entities of the application domain to be exposed as resources are models, model instances, simulations, and simulation results[2].

[2] The definitions below do not claim to be universally applicable;

Models are well-posed system models that could be simulated once all parameters are set; but the parameters are *not* set yet.

Model instances are system models that *do* have all parameters set (either explicitly or by relying on default values) and could be simulated as soon as initial conditions and input values are provided. The parameters of a model instance cannot be changed; a change in parameters always leads to a new model instance.

Simulations combine a model instance with initial conditions, input trajectories, a solver and the corresponding solver settings. They also have a state, for example `new`, `running` or `finished`, and link to their result if and only if (iff) it exists. Like model instances, simulations cannot be changed once they are created.

Simulation results contain the actual results of exactly one specific simulation. They cannot be updated either.

REST demands that each message must be *self-descriptive*, meaning that it must be actionable independent of any possible prior interaction with the same client. To support this, HTTP provides only a few methods with specified semantics and properties, for example `GET` or `POST`. In combination with the concept of resources, this means that actions which are prevalent in classical M&S environments such as Dymola need to be represented differently (compare Verborgh, Hooland, et al. 2015, section 3.3). For example, there is no such thing as a `SIMULATE` method in HTTP. Assigning a Uniform Resource Locator (URL) to an action contradicts the idea of resources and is therefore incompatible with REST. Thus, the action of starting a simulation is represented instead by `POST`ing a representation of a new `simulation`-resource to the API. Internally, the API starts the simulation as part of the handler that registers the new `simulation`-resource. Once the simulation is finished, a resource exposing the simulation result is created.

As a consequence of this design, the application state (the state of the interaction between service consumer and service instance) is only stored in the state of the resources exposed by the service, including their existence or absence. This is a desired property. However, it also means that clients need to poll the `simulation`-resource by repeatedly sending `GET`-requests in order to know about the existence of a result or the failure of a simulation (compare subsection 5.2).

Note that the developed SIMaaS-API does *not* expose the Functional Mockup Interface (FMI) functions described in the standard document (Modelica Association 2020), but more abstract/high-level functionality as outlined in Table 1.

Based on the choice of resources and a decision on how to represent actions of the application domain in terms of the addition/update/removal of resources, the service interface can be specified. Several specification formats exist, of which the OpenAPI Specification (OAS)[3] has gained widespread support. Formally specifying the service interface has several benefits: first, the service interface description serves as unambiguous documentation both for users and developers of the service. Second, parts of the service implementation can be automatically generated from the service description, such as routines for input validation, the routing of requests or a website rendering the OAS for human users. Third, test cases for verifying that the API behaves as advertised can be generated automatically from the service description.

However, relying on the OAS to specify an interface that exposes models and the ability to simulate them quickly leads to a conceptual problem: the OAS is a *static* interface description written at design time, whereas the parameters for model instantiation and triggering simulations depend on the models to be used with the SIMaaS-instance, which are only added at run time.

Three solutions to this problem suggest themselves. First, the interface description could be kept so generic that the differences between models are abstracted. This would severly diminish the advantages of using a formal service description outlined in the penultimate paragraph and is therefore undesirable.

Second, the OAS could be regenerated dynamically each time a model is added to or removed from the SIMaaS-instance. This allows the OAS to be specific enough to fully realize its potential. The translation of constraints on parameters and inputs such as maximum or minimum values or unit specifications can be automated as long as these constraints are present in the model. This approach is realized in the SIMaaS-implementation presented in this paper.

The third approach would be to *not* use a service interface description at all and let users explore the capabilitites of the server dynamically by following links. This is how human users navigate websites as they are good at finding a way to achieve their goal on a web page and understand possible consequences of clicking links without actually following them. However, this is a challenging tasks for software agents and subject to ongoing research under the term *"hypermedia API"*. The possibilities for turning the presented SIMaaS-implementation into a hypermedia API will be discussed further in subsection 5.2.

3 Implementation

Below, the software architecture and software stack chosen to realize our goal of providing a state of the art SIMaaS-implementation based on FMI as an established open standard for model export are described. For practical reasons, some requirements are posed on FMUs to

they should be seen as specific to the developed software.

[3] `https://github.com/OAI/OpenAPI-Specification`

Table 1. Overview of the service interface in terms of HTTP methods, exposed resources and their meaningful combinations.

Method	Resource	Description
`POST`	/models	Add a new model to the API-instance
`GET`	/models/{model-id}	Retrieve a model representation from the API
`DELETE`	/models/{model-id}	Delete a model representation from the API
`POST`	/models/{model-id}/instances	Instantiate a model for a specific system
`GET`	/models/{model-id}/instances/{instance-id}	Get a representation of a specific model instance
`POST`	/models/{model-id}/instances/{instance-id}/experiments	Trigger the simulation of a model instance by defining an experiment
`GET`	/models/{model-id}/instances/{instance-id}/experiments/{experiment-id}	Retrieve a representation of a specific experiment definition and its status
`GET`	/models/{model-id}/instances/{instance-id}/experiments/{experiment-id}/result	Retrieve a representation of the results of a specific simulation run

be used with the service, which are explained in subsection 3.2. In order to get the exact same simulation results from simulation of the FMU as when simulating the model in a Modelica environment, the sequence of calling the FMI methods implemented in FMPy had to be changed as explained in subsection 3.3.

3.1 Software Architecture

Figure 1 shows the high-level software architecture. Exactly one component (A) provides the service interface in the form of a HTTP-API and stores the state, in other words the resources exposed. At least one, but potentially tens or even hundreds of stateless workers (W) run simulation jobs that they pull from a task queue (Q1). The simulation results are propagated back to component A through a second queue (Q2). Both queues do not store data permanently, and neither do the workers. Requests from users do not reach the API component directly but instead arrive at a reverse proxy (R). This reverse proxy is responsible for providing an encrypted Hypertext Transfer Protocol Secure (HTTPS) connection to the outside world.

The specific software components and libraries used for implementation were chosen based on the criteria that they are well-suited for the job; Free/Libre and Open-source Software (FLOSS); represent the state of the art; and that their use avoids re-implementing functionality that already has stable implementations.

The HTTP-API (A) possibly receives many requests at once, most of which result in requests to storage or other services which are operations that are very slow compared to pure computations, meaning that significant amounts of time are spent waiting. Therefore, Node.js[4] was chosen as the programming language as it provides excellent support for non-blocking input-output (IO) operations using `promises` and the `async/await`-syntax. Also, it is commonly used for implementing HTTP-APIs and consequently offers many useful libraries that support implementation, such as the Express-framework[5] and the `openapi-backend`[6]. Incoming requests are checked for validity against their schema in the OAS. Valid requests are then propagated to the appropriate handlers, which alter or retrieve resource state and enqueue simulation requests if necessary.

The internal representation of a simulation job couples the worker implementation to the API. Workers retrieve these representations from the task queue, which is implemented using Celery[7], using RabbitMQ[8] as the message broker (Q1). As a result of only coupling API and worker through the task representation, the workers can use Python[9] as programming language. This enables the use of the pandas[10] package for representation and manipulation of time series, and allows using FMPy[11] for simulating FMUs. FMPy was chosen because it can be used natively from within a Python environment; because it is actively maintained and developed under an open-source license; and because FMPy and its dependencies can be installed easily, also as part of a container image. Worker instances can be added or destroyed according to demand and jobs are automatically distributed across all worker instances that are available. Upon finishing a simulation job, the results are propagated back to component A using Redis[12] as the result backend (Q2).

API and worker are implemented according to the *twelve-factor app*[13]-method, which is the name of a set of best practices for developing, operating and maintaining Software as a Service (SaaS).

Each component is intended to be deployed as a container. Containers are a lightweight packaging format for an application *and* all of its dependencies. They are guaranteed to run on any host that runs a compatible container

[4] `https://nodejs.org`

[5] `https://expressjs.com`

[6] `https://github.com/anttiviljami/openapi-backend`

[7] `https://github.com/celery/celery`

[8] `https://www.rabbitmq.com`

[9] `https://www.python.org`

[10] `https://pandas.pydata.org`

[11] `https://github.com/CATIA-Systems/FMPy`

[12] `https://redis.io`

[13] `https://12factor.net`

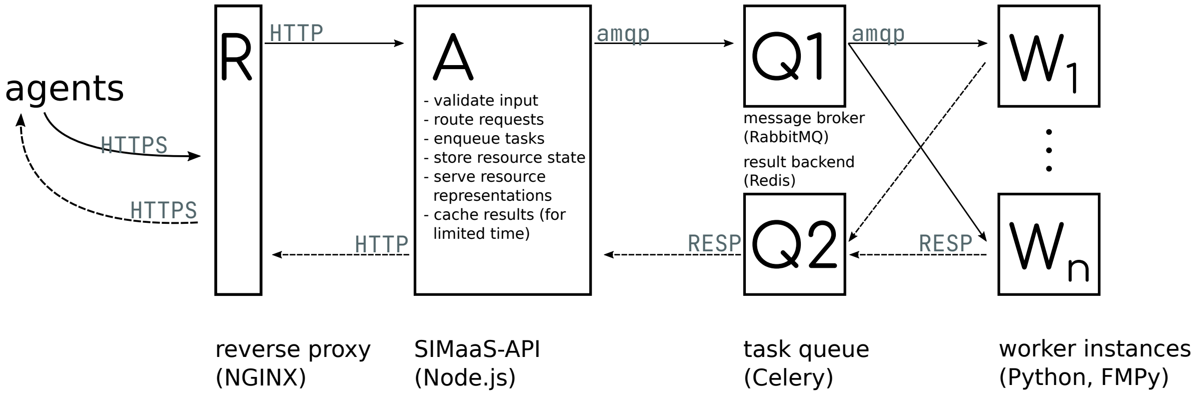


Figure 1. Software architecture of the SIMaaS-implementation.

engine, for example the Podman[14] engine. Containers are also the basic building blocks for deploying on clustered elastic platforms such as Kubernetes[15]. As such, using containers as the deployment unit for the components of the developed SIMaaS-implementation enables their use on such platforms, which in turn enables using advanced operation strategies such as automatic scaling in response to demand or load-balancing requests between several containers running the same component.

The source code for the SIMaaS-API and the workers are available subject to the conditions of the MIT license[16] at `https://github.com/UdSAES/simaas-api` and `https://github.com/UdSAES/simaas-worker`, respectively. For message broker and result backend, stock container images can be used, which are for example available on Docker Hub[17]. Consult the `README` documents in the API and worker-repositories for details.

3.2 Requirements on FMUs

FMI 2.0 for co-simulation can be seen as a way to export models *and* the corresponding solver in an open, widely supported way. In other words, a FMU can be seen as a standalone executable format of a single model using a single solver for simulation. Obviously, the capabilities and intended usage of FMI are more diverse; but for the purpose of this work, we adopt this limited view.

In order to facilitate the implementation of the envisioned software, we impose additional restrictions on the FMUs concerning their parameterization, the supported platforms for which binaries must exist, and the definition of inputs, outputs and parameters to be exposed via the API. Note that none of these restrictions impose limits on the actual models or their simulation; they merely represent a concretization of the format supported by the developed software.

Schmitt et al. (2015) investigated different possibilities to parameterize models in Dymola with respect to their subsequent export as FMU. In section 3.3 of their paper, they describe a method that “becomes favorable if the user wants to exchange whole data sets of one and the same model” (Schmitt et al. 2015, section 3.3), which is exactly the case for the SIMaaS-implementation. In short, parameters inside the model are set by inter-component references to a record. This record has a parameter `filename`, which must be set to the path of a file containing the values. All actual model parameters are set by reading this file during model initialization. This can, for example, be achieved using the `DataFiles` package distributed with Dymola or one of the functions provided in `Modelica.Utilities`.

The second requirement is that inputs and outputs of the models must be listed as such in the `modelDescription.xml` file of the FMU because the schemata for the trajectories that a service user needs to supply/can expect as a result, which are part of the OAS, are derived from this information.

Last, the FMU must contain binaries for GNU/Linux as the containers are intended to be deployed on GNU/Linux host systems.

3.3 FMI Calling Sequence

In FMI 2.0 for co-simulation, direct feedthrough is forbidden[18]. FMPy ensures this by calling the FMI functions in the order `fmi2GetXXX()`, `fmi2SetXXX()`, `fmi2DoStep()` in the `simulateCS()`-function[19].

In some cases, this leads to significant differences between the simulation results of a model simulated natively (for example in Dymola) and the simulation of the corresponding FMU. As an example, consider the simulation of a model that calculates the power generated by a photovoltaic (PV) module as a function of ambient conditions (irradiance in the horizontal plane, temperature, wind speed) and the orientation of the PV module. For this calculation, the irradiance in the horizontal plane has

[14] `https://podman.io`

[15] `https://kubernetes.io`

[16] `https://spdx.org/licenses/MIT.html`

[17] `https://hub.docker.com`

[18] Compare `https://github.com/CATIA-Systems/FMPy/issues/89#issuecomment-522949757`

[19] `https://github.com/CATIA-Systems/FMPy/blob/69ec43813e6d5f8eb79da0d17c181fe57271f8ac/fmpy/simulation.py#L1171`

to be converted to the plane of array (POA), which requires the sun's position relative to the module. At 07:00 a.m., the sun's position at 07:00 a.m. is calculated because the calculation is part of the model. At this time instant, inside the FMU, the irradiance data available is the data for 06:45 a.m. (assuming an output interval of 15 minutes) because of the calling sequence chosen by FMPy. When using Dymola to natively simulate the Modelica model, the irradiance data for 07:00 a.m. is used as intended.

For the special case of using FMUs as a portable export format of single models that are ready to be simulated using a single solver contained in the FMU, this *unnecessarily* introduces systematic errors. Therefore, we use a *fork* of FMPy which calls `fmi2SetXXX()`, `fmi2DoStep()` and `fmi2GetXXX()` in this order for the implementation of SIMaaS-workers.

4 Demonstration

The ensemble forecast for the power produced by a PV system and the search for an optimal parameter set by means of a genetic algorithm (GA) serve as examples for demonstrating the use of the SIMaaS-implementation.

The code that executes the necessary requests is written in Python in a concurrent fashion using Python's `async/await` mechanism and an asynchronous HTTP library[20]. Request sequences such as the sequence for triggering and retrieving the results of a specific simulation (`POST`ing a new simulation resource, polling its status using repeated `GET` requests, `GET`ting the result) are obviously still executed in order for each individual simulation, but they are executed in parallel for several different simulations, thereby testing/showing the ability of the SIMaaS-API to handle many requests at once.

Like the API and worker implementations, the demonstration code is available under the MIT license on GitHub: `https://github.com/UdSAES/simaas-demo`. Please refer to the `README` and the code itself for details.

4.1 Ensemble Forecast for PV Systems

A single trajectory of values such as those obtained as the result of simulating a model implemented in Modelica implies a level of exactness that does not fairly reflect the uncertainties inherent to modelling process, parameterization, and simulation.

One way to better understand and/or communicate the actual meaning of a simulation is to perform a number of simulation runs with slight variations in parameterization, input trajectories and/or initial conditions as an ensemble forecast. Ensemble forecasts created by varying the input trajectories of each simulation run are representative of situations where multiple simulations of the same model instance are required.

Here, we use the repeated simulation of a model of a photovoltaic (PV) system with the members of an ensemble weather forecast as input trajectories as an example. The PV system model is exported from the `pv-systems` library (Stüber 2020) according to the requirements on the FMUs given in subsection 3.2. The FMU is then added to the SIMaaS-instance and the request bodies to be sent for triggering the individual simulation runs are prepared by collecting the different weather forecasts. Once they are ready, all request sequences are started in parallel. Depending on the number of workers that are started, the simulations are either carried out in sequence (exactly one worker) or in parallel (more than one worker).

Admittedly, executing one ensemble forecast consisting of nine individual forecasts based on a computationally lightweight model does not require the computing resources of several nodes in the cloud. However, the advantage of using the SIMaaS-API instead of executing the FMU locally quickly becomes visible when considering that realistic real-world users for such ensemble forecasts would be utility companies responsible for stabilizing a section of the electricity grid. In this scenario, ensemble forecasts for *all* renewable energy sources in the relevant grid section would be required as part of flexibility management processes, likely to be re-calculated several times per day.

4.2 Component Selection Using a Genetic Algorithm

The second example for demonstration purposes is the search for an optimal set of values for the components of a temperature-dependent electrical circuit as shown in Figure 2, given the desired voltage at `p2` over the temperature range from $-10\,°C$ to $60\,°C$.

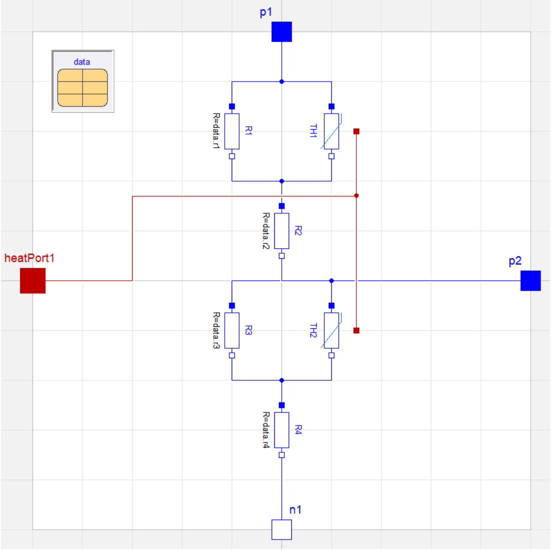


Figure 2. Thermistor network.

Suppose the resistors can each take one of the 70 values of the E24-series between 300 Ω and 220 kΩ and suppose there are nine possible values for both the resistance at reference temperature and the temperature coefficient `B` of the two thermistors. Then, there are $70^4 * 9^2 * 9^2 =$

[20] `https://github.com/aio-libs/aiohttp`

157 529 610 000 different solutions, each resulting in a different voltage over temperature-curve. A rough estimate, assuming a Central Processing Unit (CPU) time of 0.05 s per simulation, puts the total time for testing *every* permutation at around 250 years.

One possible alternative, suggested in an article on edn.com (EDN 2008), is to use a genetic algorithm (GA)[21] to search for a good solution without trying every permutation. Each possible combination of component values is seen as an individual. The fitness of an individual is evaluated by how close the voltage at p2 matches the desired voltage over the relevant temperature range, for example in terms of the Root Mean Square Error (RMSE). Because the determination of the fitness of an individual is independent of other individuals, the fitness values for an entire generation can be evaluated in parallel. After the fitness of each individual in a population is determined, the best results are recorded and the next generation is created by cross-over and mutation (subject to user-defined probabilities). The algorithm is terminated by setting a threshold for either an acceptable fitness value or a fixed number of generations.

This example represents a situation where different model instances are simulated with the same input. For finding good solutions, a few hundred simulations are likely required, many of which can be executed in parallel given a sufficiently high number of worker instances. Using a SIMaaS-instance deployed on a clustered elastic platform instead of running the GA locally becomes really beneficial iff the number of individuals in a generation is higher than the number of CPUs available locally.

Because the example is merely intended to serve as a proof of concept, no detailed analysis of the performance was carried out – neither with respect to the speed-up achieved, nor with respect to the overhead introduced by the additional software layers and the exchange of data over the network.

For implementation of the GA, the Distributed Evolutionary Algorithms in Python (DEAP) framework (Fortin et al. 2012) was used. A tiny Modelica package containing the necessary models and an example ready to be simulated in Dymola is included in the simaas-demo-repository.

5 Discussion

While the use cases described in the previous section illustrate that there are scenarios for which the developed software works and has benefits over other approaches, there are some conceptual limitations inherent to its design that should be discussed. Before summarizing these issues, we outline related work within the Modelica community.

5.1 Related Work

The concept of MSaaS has been discussed extensively in the literature. We refer the interested reader to key publications (Cayirci 2013; MSG-131 2015; Hannay, Berg, et al. 2020; Shahin, Babar, and Chauhan 2020; Kratzke and Siegfried 2020) and focus this section on previous work within the Modelica community.

Tiller (2014) motivates the use of web technologies for the design of engineering tools in general, detailing potential benefits for non-expert users. The FMQ platform and its HTML5-based interface for human users are outlined; the use of a hypermedia API as the backend of the FMQ platform is hinted at, but no details are given. The FMQ platform also uses FMUs as an executable form of a single model to be simulated using a single solver. It is a proprietary product of Xogeny, Inc.

In their 2017 presentation of the modelica.university platform, Tiller and Winkler motivate the high-level requirements for and architecture of the Aperion platform by Xogeny, Inc (presumably the successor of the FMQ platform) in addition to presenting the modelica.university website itself. With regard to concepts and the chosen technology stack, the Aperion platform seems to be very similar to the work presented in this paper, but it is a commercial product and specifics are consequently not available publicly. With regard to the use of truly RESTful technologies such as hypermedia representations and generic software clients that use them, Aperion seems to already have realized some of our plans for further work as outlined in subsection 5.2.

Bittner, Oelsner, and Neidhold (2015) outline work on a web application based on FMI 1.0 for co-simulation. Compared at a high level, the application's architecture is similar to what is presented in section 3 as there is also at least a conceptual separation between API, storage, simulation components and front-end. The provided user interface (UI) directly supports ranges for setting parameter values and seems to be intended for use by engineers. Details or source code are not readily available.

FMIGo![22] is a set of software tools for executing several coupled FMUs over the internet. It is described in a paper by Lacoursière and Härdin (2017) and available[23] under the MIT license. In contrast to the concepts of the SIMaaS-implementation explained in section 2 and the design concepts of the FMQ platform, FMIGo! choses not to make use of the design principles of the web (REST) and instead expose the capabilites of FMI withouth further abstraction through the use of (low-level) message passing protocols. Lacking the simplification of seeing an FMU as nothing but an executable form of one model with one solver, FMIGo! allows/demands chosing master algorithms for co-simulation and exposes necessary numerical details; but this clearly makes it a specialized tool for simulation experts.

Elmqvist, Malmheden, and Andreasson (2019) present the Web Architecture for Modeling and Simulation (WAMS) in terms of several use cases, the correspond-

[21] See Pelikan (2011) for an excellent introduction.

[22] https://www.fmigo.net

[23] https://mimmi.math.umu.se/cosimulation/fmigo

ing web application aimed at engineers with training in M&S and a very brief overview of its software architecture. FMUs are used for simulation using PyFMI and it is claimed that a REST API is used for exposing capabilities of the server, but the extent to which the REST constraints are realized remains unclear. It appears that WAMS is an internal project of Modelon AB.

Since version 0.2.25 (released in November 2020), FMPy features the possibility to expose the UI of FMPy including the ability to simulate FMUs as a web application. Consequently, this approach falls in line with the WAMS web application, also intended to be used by engineers, and thus caters to use cases different to those that motivate the work presented in this paper.

5.2 Results, Limitations and Outlook

The SIMaaS-implementation presented in section 3 has several desirable characteristics. First, its design reflects best practice and the state of the art for creating SaaS as identified by Kratzke and Siegfried (2020). Second, the decision to decouple API and workers, only linking them by the internal representation of tasks in the task queue, means that the implementation of how models are simulated can be changed without having to change the public service interface. This includes adding support for FMUs for model exchange or for non-FMI-based models as long as they can be expressed in terms of the resources exposed by the API (models, model instances, simulations, simulation results). For example, a user might have legacy models written in Matlab or scientific computing routines written in a general-purpose programming language which cannot be exported as a FMU. Third, the chosen software stack consists of proven and widely used open-source components.

By deploying it on Kubernetes as an example of a clustered elastic platform, horizontal scalability of the workers was shown. Graceful handling of the addition or removal of worker instances is ensured by using Celery as the task queue implementation. Implementing automatic scaling of worker instances in response to demand is only a question of properly configuring Kubernetes resources.

From our point of view, the additional restrictions posed on the FMUs to be supported by the software (subsection 3.2) are reasonable.

Using FMUs for co-simulation according to version 2.0 of the FMI standard allows using the proven solvers provided by native M&S-environments such as Dymola, but the workaround for avoiding systematic errors described in subsection 3.3 is problematic because it is not compliant with the FMI standard and because it requires maintenance work to synchronize the fork of FMPy with upstream development. Therefore, support for FMUs for model exchange and/or the upcoming FMI 3.0 standard should replace the workaround in the future.

As the presented software is predominantly research software and not a commercial product, some features that are expected of the latter are not implemented yet. For example, there is currently no access control (any user that can reach the API can send all requests); but it would be straightforward both from a conceptual and practical persepective to add this, most likely using JSON Web Tokens (JWTs). The pay-as-you-go cost model is not currently supported, but it represents mostly an organizational and operational aspect that most likely should be implemented using sidecar containers that form a service mesh (Morgan 2019) and not as part of the application code, anyway.

There exists neither an explicit threat model nor a subsequent detailed consideration of security aspects. However, basic measures against misuse were implemented. First, all requests by users are validated against the schemata defined in the OAS. These schemata are as specific as possible, including the use of regular expressions for string fields. Second, the process inside a container is run as a non-privileged user to prevent easy privilege escalation.

In contrast to the aforementioned aspects, which are of practical nature, there also exist *conceptual* limitations that keep the SIMaaS-implementation from reaching its full potential. The first limitation is that clients are required to poll resources in order to find out about changes of their state. This leads to potentially many requests that would not have been necessary and raises the question of how to set polling frequencies and eventual timeouts. Moreover, the answer to this question depends on both the current server load and the amount of workers available which cannot be known a priori. It would be possible to instruct the service to notify the consumer as soon as the resource state changes, but this would mean that the consumer has to become its own server for accepting such notifications.

The second (and more important) limitation concerns the required use of a static service interface description against which clients hardcode requests, meaning that client code will break in case the service interface changes. Dynamically re-generating the service interface description to account for the different inputs, outputs and parameters of models can, from an academic perspective, only be seen as a workaround because a static service interface description should not be necessary in the first place. Instead, clients should construct their requests at run time, based on information present in the representation received (starting at the root path of the service). This requires an alternative format for resource representation, for example the JSON-based Serialization for Linked Data (JSON-LD), because JSON lacks support for natively encoding hyperlinks (Kellogg, Champin, and Longley 2020, section 3) as well as the ability to explicitly encode the semantics of data.

Ideally, software clients should be enabled to reason about the options for advancing the application state at each step of the interaction in order to make informed decisions about which state transitions allow them to reach their goal (a desired resource state) by following links; just like humans navigate websites. The technical feasibility of this has been shown by Verborgh, Arndt, et al. (2017). Realizing such services opens exciting opportunities for building generic clients that can achieve a class of similar, yet distinct objectives while being robust against changes of the service interface as requests are assembled at run time and not constructed at design time.

We are actively working towards realizing this vision for the presented SIMaaS-implementation.

6 Conclusion

Previous work within the Modelica community to provide M&S capabilities *as a service* can be categorized in three different directions of work: providing model development and/or simulation environments as web applications in the browser for use by engineers; providing tools for distributed co-simulation; and transferring M&S functionality to the web by translating them to the architectural style REST, hoping to make use of the positive aspects of the web's design principles and cloud computing technology for M&S tooling as well as facilitating the use of M&S by software agents in a distributed setting.

The SIMaaS-implementation presented in this paper falls into the last category. It exposes models in the form of FMUs for co-simulation and the ability to simulate them as a REST-based HTTP-API that consists of an API and several worker components which exchange data using queues. Worker instances can be scaled horizontally when deployed on an clustered elastic platform such as Kubernetes.

The design concepts, software architecture and software stack are summarized and put into context by outlining related work and the achieved characteristics. Two exemplary use cases are shown. Development continues with the aim to fully support the HATEOAS principle by adding a format for resource representations that enables client-side reasoning, and to demonstrate the abilities gained by doing so.

As of this moment, the presented SIMaaS-implementation represents a functional building block for SOAs, intended to be used in combination with other services. The software is available under a permissive open-source license, readily available for testing.

Acknowledgements

This work was supported by the SINTEG-project "Designetz" funded by the German Federal Ministry of Economic Affairs and Energy (BMWi) under grant 03SIN224.

The authors would like to thank Lukas Exel and Florian Wagner for sharing their views on and experience with providing modelling and simulation *as a service*, as well as for their work on early prototypes of the presented software.

References

Bittner, Stefan, Olaf Oelsner, and Thomas Neidhold (2015-09-18). "Using FMI in a cloud-based Web Application for System Simulation". In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. Linköping University Electronic Press. DOI: 10.3384/ecp15118845.

Cayirci, Erdal (2013-12). "Modeling and simulation as a cloud service: A survey". In: *2013 Winter Simulations Conference (WSC)*. IEEE. DOI: 10.1109/wsc.2013.6721436.

EDN, ed. (2008-03-19). *Genetic algorithm solves thermistor-network component values*. URL: https://www.edn.com/genetic-algorithm-solves-thermistor-network-component-values (visited on 2021-05-08).

Elmqvist, Hilding, Martin Malmheden, and Johan Andreasson (2019-02-21). "A Web Architecture for Modeling and Simulation". In: *Proceedings of the 2nd Japanese Modelica Conference Tokyo, Japan, May 17-18, 2018*. Linköping University Electronic Press. DOI: 10.3384/ecp18148255.

Fortin, Félix-Antoine et al. (2012-07). "DEAP: Evolutionary Algorithms Made Easy". In: *Journal of Machine Learning Research* 13.70, pp. 2171–2175. URL: http://jmlr.org/papers/v13/fortin12a.html.

Hannay, Jo Erskine and Tom van den Berg (2017-10). "The NATO MSG-136 Reference Architecture for M&S as a Service". In: *Proceedings of the NATO modelling and simulation group symposium on M&S technologies and standards for enabling alliance interoperability and pervasive M&S Applications* (Lisbon, Portugal, October 19–20, 2017). STO-MP-MSG-149, p. 3.

Hannay, Jo Erskine, Tom van den Berg, et al. (2020). "Modeling and Simulation as a Service infrastructure capabilities for discovery, composition and execution of simulation services". In: *The Journal of Defense Modeling and Simulation. Applications, Methodology, Technology*. DOI: 10.1177/1548512919896855.

Kellogg, Gregg, Pierre-Antoine Champin, and Dave Longley (2020-07). *JSON-LD 1.1*. W3C Recommendation. https://www.w3.org/TR/2020/REC-json-ld11-20200716/. W3C.

Kratzke, Nane and Peter-Christian Quint (2017). "Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study". In: *Journal of Systems and Software* 126, pp. 1–16. ISSN: 0164-1212. DOI: 10.1016/j.jss.2017.01.001. URL: http://www.sciencedirect.com/science/article/pii/S0164121217300018.

Kratzke, Nane and Robert Siegfried (2020). "Towards cloud-native simulations – lessons learned from the front-line of cloud computing". In: *The Journal of Defense Modeling and Simulation. Applications, Methodology, Technology*. DOI: 10.1177/1548512919895327.

Lacoursière, Claude and Tomas Härdin (2017-07-04). "FMI Go! A simulation runtime environment with a client server architecture over multiple protocols". In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. Linköping University Electronic Press. DOI: 10.3384/ecp17132653.

Mell, Peter and Timothy Grance (2011). *The NIST Definition of Cloud Computing*. NIST Special Publication 800-145. National Institute of Standards and Technology. DOI: 10.6028/NIST.SP.800-145. URL: https://www.nist.gov/publications/nist-definition-cloud-computing.

Modelica Association (2020-12). *Functional Mock-up Interface for Model Exchange and Co-Simulation Version 2.0.2*. Tech. rep. Linköping: Modelica Association. URL: https://fmi-standard.org.

Morgan, William (2019-11-09). *The Service Mesh: What Every Software Engineer Needs to Know about the World's Most Over-Hyped Technology*. URL: https://buoyant.io/service-mesh-manifesto (visited on 2021-05-07).

MSG-131, Specialist Team (2015). *Modelling and Simulation as a Service: New Concepts and Service-Oriented Architectures*. Final Report AC/323(MSG-131)TP/608. North Atlantic Treaty Organization NATO. DOI: 10.14339/STO-TR-MSG-131. URL: https://www.sto.nato.int/publications/STO%20Technical%20Reports/STO-TR-MSG-131/$$TR-MSG-131-ALL.pdf.

OASIS (2006). *Reference Model for Service Oriented Architecture 1.0*. Tech. rep. URL: http://docs.oasis-open.org/soa-rm/v1.0/.

Pelikan, Martin (2011). "Genetic Algorithms". In: *Wiley Encyclopedia of Operations Research and Management Science*. American Cancer Society. ISBN: 9780470400531. DOI: 10.1002/9780470400531.eorms0357. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470400531.eorms0357. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470400531.eorms0357.

Schmitt, Thomas et al. (2015-09-18). "A Novel Proposal on how to Parameterize Models in Dymola Utilizing External Files under Consideration of a Subsequent Model Export using the Functional Mock-Up Interface". In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. Linköping University Electronic Press. DOI: 10.3384/ecp1511823. URL: https://2015.international.conference.modelica.org/proceedings/html/errata/errata_SchmittAndresZieglerDiehl.pdf. Revised version.

Shahin, Mojtaba, M. Ali Babar, and Muhammad Aufeef Chauhan (2020-07-24). "Architectural Design Space for Modelling and Simulation as a Service: A Review". In: *Journal of Systems and Software* 170, p. 110752. ISSN: 0164-1212. DOI: 10.1016/j.jss.2020.110752. URL: http://www.sciencedirect.com/science/article/pii/S0164121220301746.

Stüber, Moritz (2020-12-24). *UdSAES/pv-systems: v0.9.0*. Version 0.9.0. DOI: 10.5281/zenodo.4392849. URL: https://github.com/UdSAES/pv-systems.

Stüber, Moritz, Lukas Exel, and Georg Frey (2018). "Using Modelling and Simulation as a Service (MSaaS) for Facilitating Flexibility-based Optimal Operation of Distribution Grids". In: *Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics - Volume 2: ICINCO*. INSTICC. SciTePress, pp. 613–620. ISBN: 978-989-758-321-6. DOI: 10.5220/0006899106230630.

Tiller, Michael (2014). "Vehicle Thermal Management – A Case Study in Web-Based Engineering Analysis". In: *Proceedings of the 10th International Modelica Conference; March 10-12; 2014; Lund; Sweden*. 96. Linköping University Electronic Press, pp. 1073–1079. DOI: 10.3384/ecp140961073.

Tiller, Michael and Dietmar Winkler (2017-07-04). "modelica.university: A Platform for Interactive Modelica Content". In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. Linköping University Electronic Press, pp. 725–734. DOI: 10.3384/ecp17132725.

Verborgh, Ruben, Dörthe Arndt, et al. (2017-01). "The Pragmatic Proof: Hypermedia API Composition and Execution". In: *Theory and Practice of Logic Programming* 17.1, pp. 1–48. DOI: 10.1017/S1471068416000016. URL: http://arxiv.org/pdf/1512.07780v1.pdf.

Verborgh, Ruben, Seth van Hooland, et al. (2015). "The fallacy of the multi-API culture: Conceptual and practical benefits of Representational State Transfer (REST)". In: *Journal of Documentation* 71.2, pp. 233–252. DOI: 10.1108/JD-07-2013-0098.

Python Framework for Wind Turbines Enabling Test Automation of MoWiT

Johannes Fricke[1] Marcus Wiens[1] Niklas Requate[1]
Mareike Leimeister[1]

[1]Fraunhofer IWES, Fraunhofer Institute for Wind Energy Systems,
Germany,
{johannes.fricke, marcus.wiens, niklas.requate,
mareike.leimeister}@iwes.fraunhofer.de

Abstract

The development and simulation of engineering systems, especially wind turbines, is becoming increasingly complex and elaborate. At the Fraunhofer Institute for Wind Energy Systems (IWES), the in-house tool MoWiT (Modelica library for Wind Turbines) is being developed for load simulation. MoWiT is based on the modeling language Modelica and is constantly evolving. It is, thus, also becoming more and more enhanced. This results in an increased need for automation for the complex simulation setups and a need for quality assurance of simulation code used. Test automation is used to always ensure the quality of the code. The automation of various simulations and the test automation for the load simulation code are provided by PyWiT (Python Framework for Wind Turbines), which will be presented here in more detail.

Keywords Modelica, MoWiT, Python, Wind Turbines, Test Automation

1 Introduction

To represent different operating conditions of wind turbine systems, hundreds of input parameters are used for the various load simulations. Additionally, the simulation models for realistic estimation of loads and their influence on the wind turbine are getting more and more complex. This paper presents the Python Framework for Wind Turbines (PyWiT), which is developed at the Fraunhofer Institute for Wind Energy Systems (IWES). PyWiT further automates the load simulations written in Modelica using the Modelica Library for Wind Turbines (MoWiT) – also developed at Fraunhofer IWES. In addition, PyWiT is used for test automation of the load simulations to be able to automatically detect unsuitable or incorrectly implemented models and, thus, contribute to the quality control of MoWiT in an automated manner. The procedure for such a test automation is described in this paper and explained based on some examples.

1.1 Motivation

In the load simulation of wind turbines, numerous cases of environmental conditions are simulated in conjunction with different operating conditions of the wind turbines. In the design of wind turbines, for example, these include the so-called "design load cases", which comprise several hundred to thousands of simulations for a single wind turbine. In addition to normal operation under various wind conditions (speeds, profiles over the height, inclined flows, turbulence) and – if offshore – also wave and current conditions, these also include fault load cases in which the turbine shuts down under certain conditions and extreme weather conditions, such as gusts, extreme waves, and strong currents. Furthermore, to reduce dependence on chance, all simulations are run with multiple seeds for wind and waves. The combination of all these parameters results in a high number of simulations, which can only be performed with reasonable effort through automation.

All the systems in MoWiT interact with each other, which means that, when the code is changed in one model in MoWiT, there can be changes in the results for many or all the system parts at the same time.

1.2 Problem Description

As explained previously numerous combinations of input parameters (e.g., wind speed, angle of attack, seed for creating the wind fields, turbulence, etc.) are used in the load simulation of wind turbines. This results in a high three- or four-digit number of simulations for a single turbine. The combination of all these parameters by hand is therefore only possible with great effort and automation, with respect to both combinatorics and the execution of the simulations, as well as further processing of the simulation results (error checking, sorting, filtering, further evaluations) is inevitable.

Due to the strong coupling within MoWiT, even small changes or code optimizations, as well as new implementations of models, can quickly lead to different results in many parts of the entire wind energy system model. To prevent unwanted changes, e.g., due to unsuitable models or incorrect implementation, tests are implemented that automatically compare reference results

DOI
10.3384/ecp21181403

of the simulations with results of the current implementation. Thereby, the effects of changes in the code in different subsystems of the wind turbine, as well as in the different output parameters (forces, moments, generator power, etc.) can be detected and evaluated.

This paper introduces the Python Framework for Wind Turbines PyWiT and explains its structure in more detail. It is described how the program is structured, as well as how and for which practical applications, in particular for the test automation of MoWiT, it is used.

2 Material and Methods

In this section the used materials and methods are presented. After a short introduction to MoWiT, on which the framework PyWiT is based, PyWiT is explained in more detail. Beside the rough program flow, the different modular designed functions are described.

2.1 Modelica Library for Wind Turbines

MoWiT, which is available free of charge for academic use, is developed in-house at Fraunhofer IWES as a completely object-oriented simulation tool for fully coupled aero-hydro-servo-elastic simulations of wind turbines on- and offshore, with bottom-fixed or even floating substructures. It is written in the open-source object-oriented and equation-based modeling language Modelica, which can be used for various engineering systems to solve multi-physics problems. Detailed information on the development of MoWiT, as well as on the structure and components of this library can be found in the literature (Leimeister et al.2020; Leimeister and Thomas. 2017; Thomas et al. 2014; Strobel et al. 2011).

2.2 Python Framework for Wind Turbines

PyWiT, developed in-house at Fraunhofer IWES, is a program coded in Python to manage simulations of wind energy systems modeled in MoWiT. These simulations are executed in Dymola (Dymola2021). The current PyWiT version consists of six modules:

- Input Module,
- Experiment Generator (incl. Design of Experiments),
- Package Creator,
- Wind Field Generator,
- Simulation Manager, and
- Post-processing.

The single modules are organized in three main groups “Input Generation”, “Simulation Manager”, and “Post-processing”, as it is shown in Figure 1 by the simplified activity diagram. It shows the possible processing paths depending on the input point and decisions (indicated by the diamond symbols) for a single input file. A simplified presentation for one input file was chosen since the additional loops for multiple files are unnecessary for the understanding of the structure. The three main groups represent the code structure and the responsibility of the specific code. Additionally, the diagram shows the intersection between the main groups. The single groups are explained in more detail in the following subsections.

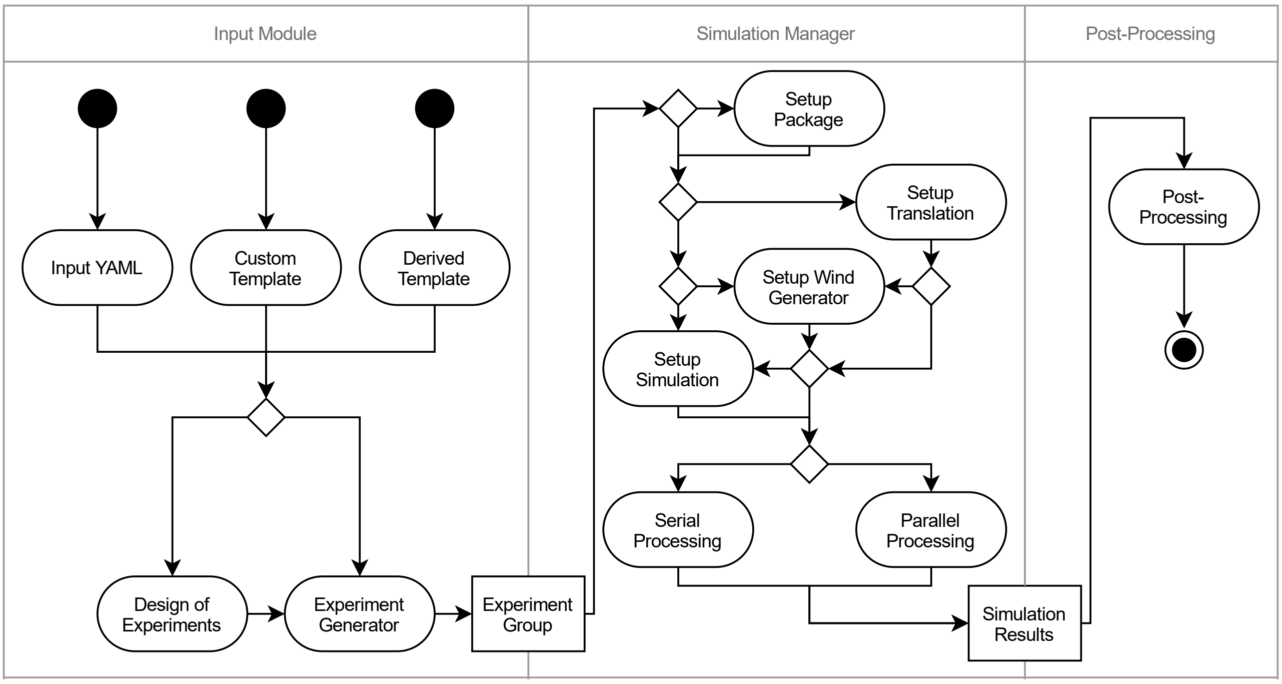


Figure 1 Simplified activity diagram for PyWiT

DOI 10.3384/ecp21181403

Input Manager

PyWiT is controlled by YAML files, which define a structured format. A structured input file, which contains all input parameters necessary for the simulations, can be created for the user without programming knowledge. PyWiT is started by specifying the input file(s) and the available processors. The input files are processed by the Experiment Generator into experiment groups, which are handled by the Simulation Manager. The Experiment Generator is responsible for sorting the data from the input file and parameterizing the input objects. Each input file results into one experiment group, which defines one set of simulations for a single model. A single simulation is represented by a data object, which contains all necessary information for independent simulation. Furthermore, there are three different types of input for the Input manager according to Figure 1. The usual YAML input specifies a fixed number of simulations by varying parameter lists. A single simulation is defined based on the common index for all lists, which leads to the requirement of equal length for all varying parameter lists. An example for the varying parameter lists is given in Listing 1.

Listing 1. Input Parameters

```
windSpeed: [9, 11, 13]

yawAngle: [-8, 0, 8]

randomSeed: [1, 2, 3]
```

This would result in three simulations, where the first configuration is given by the first element of every list. Derived templates offer simulation setups for, e.g., wind speed ranges or other commonly simulated cases.

Every input file has the option of using the Design of Experiments module. The Design of Experiments is used to expand the input files by creating the varying parameter lists from the combination of small parameter lists to obtain a large number of simulations. Currently, the Design of Experiments is based on the pyDOE2 library (PyPi.org2021) and only combinatorial designs are considered. The module offers designs, which are commonly used in wind turbine simulations. A simple example is given in Listing 2.

Listing 2. List created by Design of Experiments

```
windSpeed: [9, 9, 9, 11, 11, 11, 13, 13, 13]

yawAngle: [-8, 0, 8, -8, 0, 8, -8, 0, 8]

randomSeed: [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

The parameter lists for wind speed and yaw angle from Listing 1 are combined in a “full factorial” design, which is the combination of each list elements of the first list with all the other lists elements. This leads to nine simulations. Additionally, using a “copy-stretch” design, the random seed variable list from Listing 1 has been copied to the appropriate length. Initially, the random seed list has three elements and is therefore appended twice to itself to reach a length of nine.

Simulation Manager

The Simulation Manager takes care of the four different stages of processing for the experiment groups:

- Create Package,
- Translation,
- Generate Wind Fields,
- Simulation,

The processing of the experiment groups is separated into two steps. First, the Simulation Manager splits all Experiment Groups according to their runtime flags, so that each of the four stages described above are performed one after the other. The stages Wind Field Generation, Translation, and Simulation can be performed in parallel. The Package Creation is not parallelized, since it needs only a computing time in the range of milliseconds.

It is possible to execute these steps independently of each other if the prerequisites for the specific step are already fulfilled. For example, it is possible to perform the Translation step (possibly with the following steps) without creating a package, if a package already exists. It is also possible to create only wind fields if they are to be used for other purposes than the simulation with PyWiT. These wind fields are created by automatically writing input files for TurbSim (Bonnie J. Jonkman2009), which generates wind fields from these input files.

Furthermore, the step for creating packages implies that the dependencies of all modules containing the physical relationships of a wind turbine are determined in MoWiT and all functions necessary for the simulation of a particular model are copied together so that their dependency is maintained. This package can then be translated with Dymola, i.e., the physical interdependencies are compiled into C code and result in a translated model. This model already contains all the necessary input parameters, which, however, can be replaced by means of PyWiT to run different simulations based on the same model.

Subsequently, the physical equations are solved iteratively by Dymola. The results of each simulation are MAT files, which contain time series for all previously defined output parameters, as well as log files, which contain information about set parameters and possible errors in the simulation.

In addition to common time-series simulations, it is also possible to perform modal analyses and subsequently create Campbell diagrams.

Post-processing

The post-processing is constantly extended and includes at the current level:

- Sorting, moving, and renaming of simulation results,
- Conversion of simulation results to CSV or NETCDF data format,
- Clustering and averaging of specific time ranges for entry into an Excel spreadsheet for verification of results,
- Creating plots of different time series, probability density functions (PDFs), and power spectral densities (PSDs),
- Comparison if two time series are identical, and
- Fast Fourier transformations of different time series.

All Post-Processing modules described herein can be executed in connection with or independently of simulations. Furthermore, all these modules run independently as a stand-alone version and can be used for already existing files or new simulation results.

Test Automation

Through the various methods presented in this section, PyWiT can be used for test automation of MoWiT in addition to automatically performing simulations for further development of wind turbines. The nature of the input through YAML files allows the setup of different tests of the models created in MoWiT, for which a package is automatically generated, compiled, and simulated. Afterwards, the different Post-processing modules can be used to check the results of the test simulations and compare them with reference results if necessary. For example, it is possible to store a simulation file that has been validated as a reference in the MoWiT structure and to define this as a reference in the input file. The test model is then simulated and, first, the log files of the test simulation are checked for errors. Depending on the test, the time series of the test simulation can then be compared with the time series of the reference simulation. It is also possible to create different plots or statistical data based on the comparison. The comparison between the reference results and the new simulation results are generated automatically, but the decision whether changes to the code must be made based on these comparisons is currently the responsibility of an experienced engineer.

The test automation can either be started manually, which is useful, for example, when a new model is written in MoWiT before it is merged from the develop branch into the master branch. In addition, it is possible to run certain tests automatically on a regular basis (under Windows, for example, through the task scheduler) to obtain regular information about the quality of MoWiT.

3 Results

As described in the previous section, PyWiT offers extensive possibilities for automating simulations and various tasks around these simulations. Through the Design of Experiments, it is possible to create various combinations of input parameters by minimal user input and to run simulations with them. Different Post-processing modules can be used to evaluate the simulations on the one hand and to test the MoWiT code on the other hand. Thus, PyWiT extends the existing MoWiT tool and automates otherwise labor-intensive and error-prone tasks, leading to higher quality results and reduction of working time and, hence, costs in wind turbine development.

An example of an evaluation from test automation is shown in Figure 2.

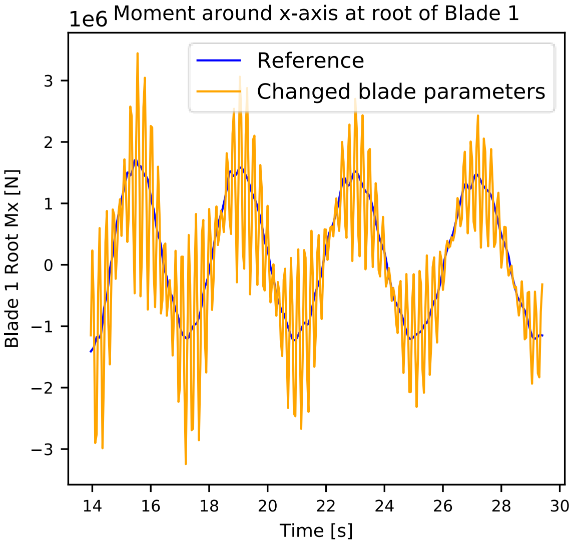


Figure 2 Comparison of the moment around the x-axis on a blade of a reference model and a model with shifted aerodynamic axis in the blade

Here, two models with identical input parameters were compared. They differ in the position of the aerodynamic axis in the blades of the wind turbine, which has an influence on almost all result variables. As an example, the moment in the root of a blade is shown here. In this case, the blue line is considered the reference model, which means that those results come from a valid simulation. The orange line is compared to that valid simulation, to see if the code still works reasonable. The plot is created automatically, while the interpretation of the results is made by hand. The mean values of the two values only have small differences, but the changed aerodynamic axis leads to higher-frequency oscillations that are superimposed on the reference signal. Such a figure, or one like it, can be evaluated by an experienced engineer to determine if there are errors in the code and where they may lie.

4 Discussion

The presented PyWiT framework eases the setup of various simulation studies. Adjustments of simulation setups are simplified and sped up by taking advantage of object-oriented programming methods for the definition of simulation cases. This is especially useful in the setup of simulation series according to standards, e.g., (IEC2019; DNV GL. 2016). Along with the specific standard, the simulation series can be defined by derived YAML templates from class instances, which are initialized by specific wind turbine parameters. Common variable definitions are distributed to different simulation cases by class inheritance. Specific wind conditions can be implemented as single classes and used as modules in the simulation series class.

Furthermore, high flexibility can be maintained by implementation in individual packages. By the separation of Experiment Generator and Simulation Manager the coupling between code blocks is reduced. This makes code adjustments more efficient. The Experiment Generator produces single experiments, which can be analyzed and thereby the errors in the setup are reduced. An example would be the settings for wind conditions like the wind shear. They can be specified for the generated wind fields or directly in MoWiT but should be only set in one of these options. The Design of Experiments for this framework is carried out often for variables with more than two levels. Since a full factorial design leads to a high number of simulations, a combinatorial design offers a possibility to reduce the simulation amount. However, a combinatorial design could lead to bias in a simulation series, when correlating influences are combined in the design.

Implementation of a simulation framework can be a complicated task. Our example shows one way of structuring the required steps from the simulation definition to the execution. A guideline for other developers/researchers would be to develop code, which can be changed without much effort. The reduction of coupling in the code, aim for high flexibility, and coding principles like DRY (Wilson et al.2014) are helping to achieve that goal.

5 Outlook

PyWiT is already at a stage of development where it can take over many tasks automatically; however, it is still in constant further development. In addition to various modules for Post-processing, also for the extension of test automation, the applications Load Simulation Verification, Distributed Computing with HTCondor, and Optimization, which will be explained in the following, are already ongoing or planned. At the end, further planned advancements are briefly discussed.

5.1 Load Simulation Verification

Load Simulation Verification is a methodology, applied at Fraunhofer IWES (Huhn and Popko2020), for comparing different load simulation codes, such as FAST (Jason Jonkman2018; DNV GL, 2018) or Bladed (DNV GL, 2018; DNV GL, 2018), etc.), with MoWiT. In addition to various plots with time series, PSDs, and PDFs, the focus here is on an XLSX file in which various sensors are available for different input parameters. This XLSX file is reviewed and evaluated by an experienced engineer and any deviations in the simulations are evaluated.

An automatic execution of the simulations in MoWiT for the verification and a subsequent creation of the different plots, as well as the filling of the XLSX file, is in the alpha version and is currently being tested in different projects.

5.2 Distributed Computing with HTCondor

In order to efficiently use computing resources, which are often distributed over several computers, a distributed computing system has been developed at Fraunhofer IWES based on the open-source software HTCondor (HTCondor2021). This system distributes the simulations to the available computing resources and manages the simulations that are entered into the queue system by different users. The already tested system can easily be extended by further computing resources. An integration of the system for distributed simulation in PyWiT is currently under development.

5.3 Optimization

The characteristic of PyWiT of automatically executing tasks facilitates its usage for mathematical optimization for a broad range of problems. One can make use of the various existing open-source optimization libraries in Python. Optimizing wind turbine parameters was already possible and extensively used in a previous version of the framework (Leimeister. 2019; Leimeister et al.2021). These range from the design optimization of the entire wind energy system or specific components, such as the floating support structure, to controller tuning optimization tasks or further applications. The objectives are mostly related to cost minimization, load reduction, or performance improvement (Leimeister et al.2020; Leimeister et al.2020), but can also address system or component scaling (Leimeister et al. 2019), as well as reliability-based design optimization targets (Leimeister and Kolios2021).

A major advantage of PyWiT for the further use for solving optimization problems is the modular design. Various simulations with different parameters used as optimization parameters can be combined with various objective functions and constraints which are defined in the Post-processing module. Therefore, holistic and multi-disciplinary applications are possible through combination for realizing any kind of optimization problem, e.g., multi-objective optimization of different components or consideration of fatigue and extreme loads.

As for each optimization problem and corresponding considered system certain optimization algorithms are more suitable than others, the large number of different optimization platforms, tools, and algorithms must be exploited when aiming for solving diverse optimization problems. However, each optimization algorithm has certain parameters and options to be specified. Therefore, a modular and standardized interface to different Python optimization libraries is currently developed at Fraunhofer IWES. It allows the easy use of different optimizers by utilizing the same input definitions for the optimization problem, comprising optimization parameters, objectives, and constraints. PyWiT will be connected to the interface in future.

5.4 Further Planned Advancements

Further advancements will include the fully automated creation of different design load cases in a separate module. The module will make use of the Design of Experience module and contain the required parametrization of turbine models and simulation setup. This will replace the current template-based approach. Another step includes further implementations for Post-processing (DNV GL, 2018), which partially already exist in separate tools. These comprise, among others, fatigue load evaluation, using Rainflow Counting and Miner's rule, as well as other load evaluation methods.

6 Conclusions

In this paper, the Python Framework for Wind Turbines PyWiT developed at Fraunhofer IWES is described. It was shown that PyWiT already automates many tasks around the simulation of wind turbines and, thus, makes the development of wind turbines faster and cheaper. Besides the automated input of many parameters by the Design of Experiments, PyWiT can also be used for test automation of the simulation code MoWiT to guarantee the quality of the code. PyWiT is already used for many tasks at Fraunhofer IWES, including load verification, but is under constant development, especially in Post-processing.

References

Bonnie J. Jonkman (2009) "Turbsim User's Guide." Technical Report. *National Renewable Energy Laboratory.*

DNV GL (2016-10) *Loads and Site Conditions for Wind Turbines (Standard DNVGL-ST-0437).* November. DNV GL AS, www.dnvgl.com/. Accessed 9 Oct. 2018.

--- (2018-01) *Bladed Theory Manual: Version 4.9.* 1 Jan. 2018.

--- (2018-01) *Bladed User Manual: Version 4.9.* 1 Jan. 2018.

Dymola: (Dynamic Modeling Laboratory) (2021). Dassault Systèmes, 2021. Accessed 16 Apr. 2021.

HTCondor: High Throughput Computing (2021). University of Wisconsin, Madison, USA, 2021. Accessed 16 Apr. 2021.

Huhn, Matthias L., and Wojciech Popko (2020) "Best Practice for Verification of Wind Turbine Numerical Models." *Journal of Physics: Conference Series,* vol. 1618, p. 52026. doi:10.1088/1742-6596/1618/5/052026.

IEC (2019-02) *Wind Energy Generation Systems - Part 1: Design Requirements (International Standard IEC 61400-1:2019-02).* 4.0th ed. International Electrotechnical Commission. International Standard.

Jason Jonkman (2018-01) *NWTC Information Portal (FAST).* 1 Jan. 2018, nwtc.nrel.gov/FAST. Accessed 21 Apr. 2021.

Leimeister, Mareike (2019-03) "Python-Modelica Framework for Automated Simulation and Optimization." *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6,* 2019, March 4-6, 2019, Regensburg, Germany, The 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019. Linköing University Electronic Press, 2019pp. 51–58. Linköping Electronic Conference Proceedings.

Leimeister, Mareike, et al. (2020) *A Fully Integrated Optimization Framework for Designing a Complex Geometry Offshore Wind Turbine Spar-Type Floating Support Structure.*

Leimeister, Mareike, and Athanasios Kolios (2021) "Reliability-Based Design Optimization of a Spar-Type Floating Offshore Wind Turbine Support Structure." *Reliability Engineering & System Safety,* vol. Document 32009L0028, no. 4, p. 107666. doi:10.1016/j.ress.2021.107666.

Leimeister, Mareike, Athanasios Kolios, and Maurizio Collu (2020) "Development and Verification of an Aero-Hydro-Servo-Elastic Coupled Model of Dynamics for FOWT, Based on the MoWiT Library." *Energies,* vol. 13, no. 8, p. 1974. doi:10.3390/en13081974.

--- (2021) "Development of a Framework for Wind Turbine Design and Optimization." *Modelling,* vol. 2, no. 1, pp. 105–28. doi:10.3390/modelling2010006.

Leimeister, Mareike, Athanasios Kolios, and Maurizio Collu, and Philipp Thomas (2019-06) "Larger MW-Class Floater Designs Without Upscaling? A Direct Optimization Approach." *ASME 2019 38th International Conference on Ocean, Offshore and Arctic Engineering: Volume 1: Offshore Technology; Offshore Geotechnics*, June 9-14, 2019, Glasgow, Scotland, UK, ASME 2019 38th International Conference on Ocean, Offshore and Arctic Engineering. American Society of Mechanical Engineers, 2019.

--- (2020) "Design Optimization of the OC3 Phase IV Floating Spar-Buoy, Based on Global Limit States." *Ocean Engineering,* vol. 202, no. 1, p. 107186. doi:10.1016/j.oceaneng.2020.107186.

Leimeister, Mareike, and Philipp Thomas (2017-05) "The OneWind Modelica Library for Floating Offshore

Wind Turbine Simulations with Flexible Structures." *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, May 15-16, 2017, Prague, Czech Republic, The 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017. Linköping University Electronic Press, 2017pp. 633–42. Linköping Electronic Conference Proceedings.

PyDOE2 1.3.0 (2021-04) PyPi.org. 1 Jan. 2021, pypi.org/project/pyDOE2/. Accessed 20 Apr. 2021.

Strobel, M., et al. (2011-03) "The OnWind Modelica Library for OffshoreWind Turbines - Implementation and First Results." *Proceedings from the 8th International Modelica Conference, Technical Univeristy, Dresden, Germany, March 20-22, 2011*, March 20-22, 2011, Dresden, Germany, The 8th International Modelica Conference, Technical Univeristy, Dresden, Germany, March 20-22, 2011. Linköping University Electronic Press, 2011pp. 603–09. Linköping Electronic Conference Proceedings.

Thomas, Philipp, et al. (2014-03) "The OneWind Modelica Library for Wind Turbine Simulation with Flexible Structure - Modal Reduction Method in Modelica." *Proceedings of the 10th International Modelica Conference, Lund, Sweden, March 10-12, 2014*, March 10-12, 2014, Lund, Sweden, the 10th International Modelica Conference, Lund, Sweden, March 10-12, 2014. Linköping University Electronic Press, 2014pp. 939–48. Linköping Electronic Conference Proceedings.

Wilson, Greg, et al. (2014) "Best Practices for Scientific Computing." *PLoS biology,* vol. 12, no. 1, e1001745. *Guidelines for effective coding. Write programs for people, not computers. Let the computer do the work. Make incremental changes. Don't repeat yourself (or others). Plan for mistakes. Optimize software only after it works correctly. Document design and purpose, not mechanics. Collaborate.*, doi:10.1371/journal.pbio.1001745.

A Graph-Based Meta-Data Model for DevOps in Simulation-Driven Development and Generation of DCP Configurations

Stefan H. Reiterer[1] Clemens Schiffer[1]

[1]Department E, Virtual Vehicle Research, {stefan.reiterer,clemens.schiffer}@v2c2.at

Abstract

To improve the quality of model-based development and to reduce testing effort DevOps practices gain more and more importance. However, most system engineers are not DevOps specialists and there are a lot of manual steps involved when writing build pipelines and configurations of simulations. For this purpose, an abstract graph-based meta-data model is proposed. This allows auto-generation of scenario descriptions for the DCP standard and code for the build server where the simulation is set up and executed. A simple use case is described as an example of how this could be applied in practice. Furthermore, a Python implementation of a DCP master and a simple FMI to DCP wrapper are presented in this work.

Keywords: Continuous Integration, DevOps, MBSE, NoSQL Graph Data Bases, DCP, SysML, UML, SSP

1 Introduction

To tackle the growing complexity of software on electronic control units (ECUs) in cars or co-simulation of physical phenomena of different parts of the vehicle the use of practices from software development is on the rise. Especially DevOps plays an important role. According to Bass, Weber, and Zhu (2015) DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into production while ensuring high quality.

However, while DevOps is well established in software development there remain a few challenges when simulations come into play. Simulations are often very complex and need a lot of expert knowledge from other fields like mechanical or electrical engineering. Thus, there is a need to provide frameworks which need only little knowledge of DevOps tools like build servers to enable the application of proper development practices.

Furthermore, when performing model-based engineering there may be several components available from previous or parallel projects. Thus, it would be convenient to integrate them in the current workflow and test the system in several variations (models, versions, or different parameters). Hence, it would be more efficient to allow automatic setup of existing artifacts and pipelines from an abstract description provided in UML, SysML or the SSP standard and then generate the necessary simulation setup from that description, at least in a semi-automatic fashion. For this, the Distributed Co-Simulation Protocol (DCP) standard was chosen since it allows abstract description of co-simulation configuration for very different kinds of setups. In Section 2 a simple use case is provided as an illustrating example.

To achieve this goal an abstract graph data structure is introduced in Section 3 to build the link between system engineering tasks, DevOps and co-simulation. Furthermore, it is shown that the data structure is suited for data transformations between general scenario descriptions and co-simulation scenarios and we will discuss the implementation of the use case in more detail in Section 4.

2 A Simple Use Case

In this section a typical process in development of simulation models and the roles involved are described. A use case for this process is derived. The challenges and potential benefits of applying dedicated DevOps methods are highlighted.

2.1 Use Case Description

The following scenario is considered. A system engineer wants to test two related software components. Their common behavior defines a system, which is subject to usage in product development projects at a later point in time. They know that there are two prototypes from development available but they want to rely on the nightly version to use the most up to date version. They want to experiment and incorporate small changes, hence the software components have to be build from scratch in a regular fashion. To use them, the system engineer has to

- get access to the code,
- build a pipeline (or script) and
- set up a co-simulation scenario and run it.

See Figure 1 for a schematic of the use case. The content of the red box highlights what the system engineer has to define. The code, shown in green boxes, is maintained by developers. The boxes in blue are related to process automation. Typically, a DevOps engineer is responsible for implementing these activities. A clear separation of tasks enables every member of the team to focus on their respective role in the process.

DOI
10.3384/ecp21181411

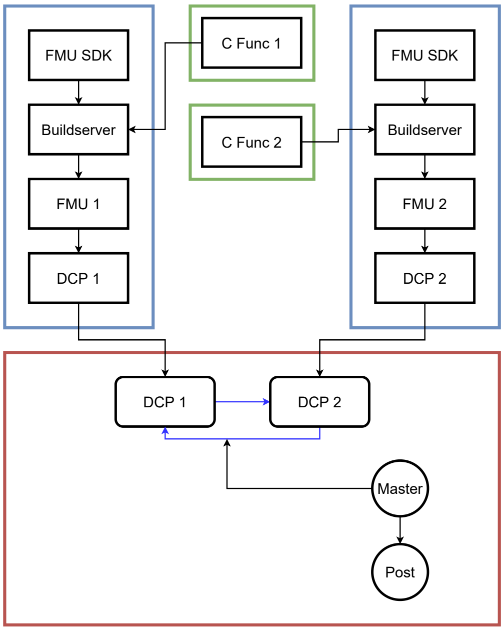


Figure 1. Schematics of a Simple Build and Simulation Pipeline

- The system engineer defines the system designs including model boundaries, their scope and in particular the flow of model signals between models. This can either be done from scratch or by working in part with previously established models.
- The model developer creates the models and their implementation according to the previously defined system design.
- The DevOps engineer provides build pipelines – or templates for build pipelines – to build the models and deploy the resulting instances of these models as artifacts.

Similarly pipelines for running the simulation need to be provided. These pipelines should be as flexible as possible to be parametrized for different configurations and situations. In Figure 2 the description of the simulation participants is depicted. Here as an example UML was used for the proof of concept. However, this is not limited to UML since other standards like SysML or SSP could be used for the transformation as well since they are easily parsable.

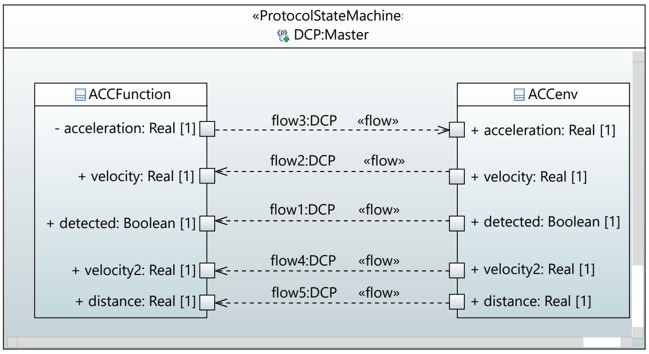


Figure 2. System described in UML

2.2 Challenges

Although the tasks described in the previous subsection are manageable in general, they suffer from several well-known problems. First of all, the entire process is considered complex, and involves many different tasks as sources for faults and errors. Fixing problems is time consuming and unnoticed errors can lead to disasters. Especially setting up the infrastructure for builds is not always straight forward and needs proper configuration management. This includes setup of scripts, pipelines, specific software versions, etc. This may be an additional challenge for someone with little software engineering background. Even with the help of the developers and DevOps engineers it may be cumbersome. Staff might not be available all the time. Reaction time might be limited, so several of the arising issues may not be fixed immediately. For these reasons it would be desirable to have dedicated mechanism in place. This paper contributes by (1) introducing a method for setup of simulation-driven development processes that rely on graphs, (2) provision of an implementation consuming these graphs, automating the build process, and generate prototypical systems for simulation and testing.

3 Co-Simulation Process Graphs and the DCP Standard

In this section the idea of using a process graph for simulation execution is elaborated. The used co-simulation standards are presented and the specific challenges of using graph-based DevOps methods for configuration management are identified. An theoretical overview of the implementation is given.

3.1 Motivation

There are several data formats and tools available to tackle the tasks for the systems engineer described in the previous section. However, the main problem for modern engineers are the interfaces and steps which are necessary to go from one domain to the other. Co-simulation protocols like DCP on the one hand solve this problem from the view of simulation by providing uniform interfaces and descriptions of the simulation participants. Graph-based workflow descriptions on the other hand allow setting up continuous integration pipelines, as discussed e.g., for Gitlab CI in (Pundsack 2018). Nevertheless, when dealing with simulations there are problems. Existing graph-based solutions for workflow descriptions either rely on structural properties like having no directed cycles (directed acyclic graphs or DAGs for short) or only work in an online set-

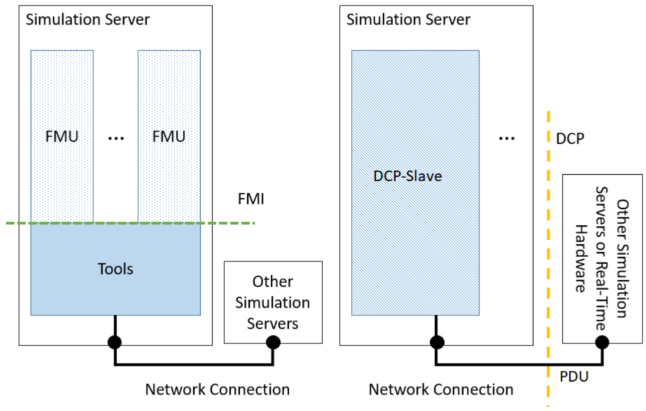


Figure 3. Comparison FMI-based simulation and DCP-based Simulation

ting. This stems from the fact that orders of workflows can only be described by a graph if there are no cycles, but when dealing with closed loop simulations such cycles are introduced naturally. It is beneficial for optimization, analysis and resource management tasks and automatic code generation of build scripts to provide a data structure which works offline and still describes all dependencies and all communication ways.

3.2 Co-Simulation Standards

The FMI standard has greatly simplified the exchange of simulation models across tool boundaries (Blochwitz et al. 2012). However, a corresponding standard for real-time systems that includes network communication is missing. The Distributed Co-Simulation Protocol (DCP) is an application-level communication protocol, designed for cost-effective development processes and opportunities to easily integrate models into simulation environments (Krammer and Blochwitz 2018). It standardizes the exchange of simulation-related configuration information and data (Modelica Association Project DCP 2019). The DCP standard fills this gap of FMI by standardizing the behavior of the model and the exchanged messages on the level of the communication protocol to simplify the integration of different real-time systems. It reduces the configuration effort required drastically thereby increasing the efficiency of tests and simulations. See Figure 3 for a comparison of the two standards, for FMI see (FMI-Working-Group 2020, p.97). Furthermore, the DCP standard defines a way to describe scenarios by providing a specified XST tranformation for scenarios which can be directly used for the automatic configuration of the simulation setup (Krammer and Benedikt 2018). Hence it is perfectly suited for the goals of the use case from the foregoing section, and we can use it as leverage for the automation of the whole workflow.

3.3 The Co-Simulation Process Graph

To use the leverage of the tools available without introducing a new software chain a graph-based meta-data model was introduced which allows the description of workflows and simulation scenarios in a unified manner. The so-called co-simulation process graph was formally defined in (S. H. Reiterer et al. 2020) and is an extension to the classical process graph (Tick 2007). It solves the problem of cycles introduced by closed loop simulations and models without the need of separating the workflow sequence and the topology of the simulations. By definition a co-simulation process graph is a directed graph with the following properties:

- The set of nodes consists of data nodes, transformation nodes, master nodes, signal nodes and communication (or gateway) nodes.
- To represent the instantiation of a process or the usage of a signal inside a simulation, copies of the nodes which represent these instances are made. Instances have to be directly connected to their originals.
- Instead of using the bi-partite structure to represent data transformations, only instances of processes can connect to data nodes to perform operations. In this way, the nodes which perform operations and their instantiation can be determined with a suitable algorithm, which determines a different partition of the graph with help of the defined structure, to provide the correct order of executions. This is necessary since it is allowed that transformation nodes are neighboring, e.g., a Docker container which is built and then used for executing a program afterwards.
- An information node can never be the successor or predecessor of another information node. A process must be placed in between. However, neighboring process nodes are allowed. This may happen if a program-performing transformation at a later stage is modified beforehand by another process (e.g., parameterization of tools).
- A simulation is a sub-graph with the following properties: a) It contains the instance of a master node. b) The instance of the master node is connected to all instances of signal nodes that belong to the simulation. c) All the other nodes inside the simulation (i.e. the simulation participants and communication gateways) neighbor a signal instance. d) Each instance of a signal is only allowed to appear once inside a simulation.
- Cycles are only allowed inside a simulation subgraph.

A more detailed description of the data structure and analysis of the used algorithms can be found in the paper (S.

Reiterer and Kalab 2021), which was recently accepted in the International Journal of Simulation and Process Modelling. An example is shown in Figure 4. A possible example is that the nodes c_1 and c_2 represent software sources (e.g., source code of a model) b represents a build tool like CMAKE and b_1 and b_2 represent two processes of this build tool which are started, which leads to the simulation units P_1 and P_2, while the node M represents a simulation master. After the build in stage 1) the simulation is executed and the master is configured by the information contained in the node M and gets additional parameters from node I, while the node O represents the output of the simulation. The nodes i_j and o_j represent in- and outgoing signals like velocity or acceleration, while g_j represents the communication protocols (e.g., a network protocol like IP) for $j = 1,2$.

It can be shown that a co-simulation process graph can efficiently be transformed into a DAG. Those transformations are based on contractions which are of particular importance since they allow a simplification of the graph to make the representation more accessible for human users, because the data structure was designed to be computer friendly and can become quite complex.

The transformation nodes can also be filled with existing scripts and code snippets to improve reusability of existing build scripts. This way the graph structure can be either directly used as a simple low code platform for programming pipelines or linked with existing technologies to make the combination of the continuous integration world with the realm of simulations easier. Since it is a universal data structure it is not necessary to introduce new tooling but allows linking the existing tools into the framework.

See Figure 5 for the database view on the build graph for the adaptive cruise control (ACC) function as a further example. This graph contains all necessary steps for the build steps of the ACC function participant. Additionally, the database holds the signals which are associated with the participant which should be used in the scenario description for the DCP simulation. Several manufacturers have fixed catalogs of signals which are allowed to use.

Storing them in the database together with simulation participant helps developers to avoid using wrong signals.

4 Implementation

In this section the details of the implementation are presented. This includes the description of the used simulation participants and how the co-simulation standards are used. Further the configuration management using metadata embedded in the process graph and the deployment of the resulting configuration to the simulation are discussed.

4.1 General Implementation

The prototype of the graph transformation service was implemented in Python with usage of the networkX library (Hagberg, Schult, and Swart 2008). The transformation service transforms the graph description of the build pipeline from XML and stores it in a graph database. Here ArangoDB was used due to the generous license model for the community edition (BSD like license) and since it is recommended as one of the stronger candidates (Fernandes and Bernardino 2018). Furthermore, the scenario description was drawn with the Eclipse plugin Papyrus since it is one of the few free SysML/UML modeling tools available which allow export of the UML diagrams as text file (Lanusse et al. 2009). As a build server Jenkins was used for which the build pipelines were generated. To ease the deployment Docker containers were built in which the simulations could run without worrying if the necessary libraries are available on the Jenkins slave which performs the build process.

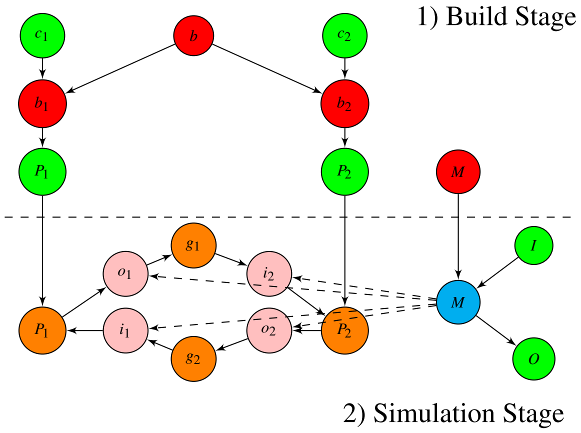


Figure 4. Simple Example of a Co-Simulation Process Graph

4.2 Implementation of the Use Case in Detail

From the abstract description depicted in Figure 2 a simple XML parser in Python was used to transform the model exported from Papyrus to generate the necessary nodes and edges for the simulation subgraph (in the sense of Subsection 3.3) and the provided meta-information which was entered into the description is parsed and entered into the nodes.

The service then sends AQL (Arango Query Language) queries to the graph DB and gets the related build pipelines for the simulation participants and the master and combines them. After this the DCP description file is generated from the graph description directly on the build servers' workspace where the simulation is executed. Hereby the graph service enters all connections between the participants and the configuration of the master. Furthermore, the signals with the used value references are taken from the database. After that all subgraphs for the pipelines are connected and the necessary code for the build server is generated and executed. See Listing 1 for the generated code. In lines 4-13 the setup of the participants is executed. The necessary commands and parameters for the setup are stored within the graph database. In lines 14-18 a simplified configuration file is written which represents the signal connections via reference values. However, this configuration is for demonstration only. Every line in the configuration file config.ini represent the connection of one output to one input referred to by the corresponding participant name and value reference. For demonstration purposes we simply write these lines one by one by diverting the output the echo commands to the file. Note that these commands were automatically generated from the structure of the simulation graph. In a production environment such configuration data has to be supplied in a structured manner, such as an XML-file, a prototype which uses an advanced scenario description format exists as well. Finally in line 20 the simulation is started; this is achieved by running the dcpmaster-service in its own docker. This service in turn uses the generated configuration file to configure and start the simulation participants, which were started up before and were waiting in an idle state for commands.

This pipeline can be subdivided in a build stage and a simulation stage as depicted in Figure 4. During the build stage the simulation participants are built. Two models are to be simulated in the system: An adaptive cruise control (ACC) function which controls the acceleration of a vehicle based on the speed of the vehicle and the distance to and speed of a vehicle ahead, if such a vehicle is detected. This ACC functionality is tested by coupling it with a second model that simulates the entire environment of the ACC, including the ego-vehicle and its surroundings, refer to Figure 2 for the signal flow. Both models are implemented as C code with accompanying meta-data. A build pipeline template uses this code and meta-data from a repository to build the models with their dependencies resulting in finished artifacts, in this use case two FMUs. These artifacts are then tested – both with respect to formally complying to the FMI standard as well as their basic functionality – and subsequently uploaded to an artifact repository.

In the next step of the build stage the simulation environment is set up. Docker containers are build in which the simulation participants will run. A DCP wrapper for FMUs was implemented based on the DCPLibrary – the open-source reference implementation of the DCP. This is used to the make functionality of each FMU available in a DCP slave in a distributed FMI master configuration. This is possible because the DCP standard was designed with the goal of basic comparability with FMI in mind. Such a wrapper can be used to integrate existing high-quality FMU in a networked DCP-simulation or real time system. During the simulation stage the simulation is configured and executed. The docker containers – one for each simulation participant – are started. The configuration of the scenario is generated from the meta-data of the simulation graph, the lower part of Figure 4, this includes which input is connected to which output, parameters and the timing configuration. This configuration is provided to the DCP master, which deploys this configuration to the slaves and starts the simulation. In the use case the DCP master is implemented in Python, demonstrating the interoperability of DCP implementations. A feature was implemented that reads in a DCP scenario description, containing slave descriptions of all DCP slaves involved in the scenario and the desired configuration details, and can generate from this the necessary configuration sequence to roll out the configuration and run the simulation. The master controls the simulation with DCP protocol data units (PDUs) that are sent via network connections to the DCP slaves. The DCP slaves in turn exchange simulation data via data PDUs until the master stops the simulation.

Listing 1. Generated Code for Simulation

```
1  stage(’Stage: Sim77706 77706,Sim888’) {
2  steps{
3        sh ’echo "Run Simuation Sim888"’
4        script {
5                DockerFolder = "
                     dcp_docker_run"
6                Key = "part1"
7                sh "cd ${DockerFolder}/;
                     docker-compose up -d $
                     {Key}"
8                }
9        script {
10               DockerFolder = "
                     dcp_docker_run"
11               Key = "part2"
12               sh "cd ${DockerFolder}/;
                     docker-compose up -d $
                     {Key}"
13               }
14       sh ’echo "ACCenv, 6, ACCFunction,
             4" >> config.ini’
15       sh ’echo "ACCenv, 3, ACCFunction,
             2" >> config.ini’
16       sh ’echo "ACCenv, 4, ACCFunction,
             3" >> config.ini’
17       sh ’echo "ACCenv, 1, ACCFunction,
             1" >> config.ini’
18       sh ’echo "ACCFunction, 5, ACCenv,
             7" >> config.ini’
19       sh "cp config.ini dcp_docker_run/
             dcp_py_master/ACC/"
20       sh "cd dcp_docker_run/; docker-
             compose up dcpmaster"
21          } }
```

The simulation results are then post-processed with a Python script, zipped and put on an artifact repository, e.g., JFrog Artifactory, where they can be downloaded. In Figure 6 an overview of the generation and linking procedure is provided.

5 Conclusion and Outlook

In this work we presented a simple proof of concept to showcase how CI pipelines could be described in the context of simulations. For this an abstract graph model was described which can be stored within a database and can be used for splitting work and autogenerate simulation scenarios out of model descriptions. A simple simulation was described in UML which was used directly for generating the simulation configuration within the pipeline. Furthermore, we demonstrated how the described pipelines can be stored within a graph database

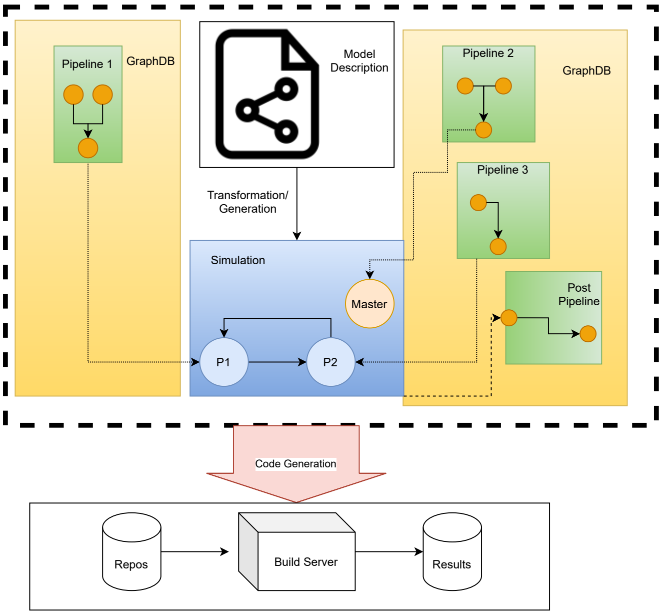


Figure 5. Transformation of Model and Linking of Subgraphs

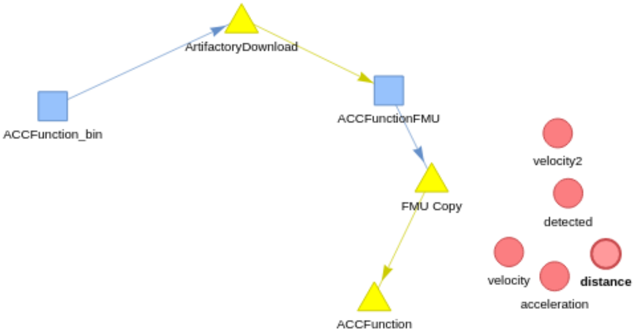


Figure 6. Database View of the ACC Function Build-Graph

and how it can be used to merge existing pipelines to a complete workflow containing all necessary steps. For the simulation part we demonstrated how FMU participants can be converted to DCP participants and how to roll them out automatically via Docker containers within the generated pipeline. Connections between the participants are automatically set with help of the graph data structure and participants are automatically started with help of the provided meta-data. However, it is important to highlight that this was a mere proof-of-concept. There are a lot of aspects regarding proper simulation description which still are open. Currently only a primitive mapping was used but there are several aspects regarding system design and work processes which were not considered yet. Extensions for SysML and SSP are planned to provide a proper tool for systems engineering. Since the proposed graph data structure is very abstract and very flexible there are also quite a few topics which have to be investigated on the implementation side. One topic is the algorithmic analysis of the process graph; partial results were presented on the Grazer Symposium of the Virtual Vehicle (S. H. Reiterer 2020). As already mentioned in Section 3.3 a companion journal paper was recently submitted and accepted (S. Reiterer and Kalab 2021). In this work the algorithmic and modeling aspects of the co-simulation process graph model are described in detail. Furthermore, strategies to optimize the resulting pipelines are investigated. Another big topic is the proper integration of the graph database. This includes versioning of graph-based pipelines within the graph database and managing of configuration parameters and tool variation. Regarding the DCP standard there are several questions remaining like defining a comprehensive standard for the description of the co-simulation process graph elements (node and edge data) to ensure generation of DCP scenario descriptions in a consistent manner. This would not only open the possibility to describe DCP simulations as graphs but would also enable computer aided analysis of the performance of simulations on a large scale or automatically deploy different variations of scenarios.

Acknowledgements

This publication was written at Virtual Vehicle Research GmbH in Graz, Austria. The authors would like to acknowledge the financial support within the COMET K2 Competence Centers for Excellent Technologies from the Austrian Federal Ministry for Climate Action (BMK), the Austrian Federal Ministry for Digital and Economic Affairs (BMDW), the Province of Styria (Dept. 12) and the Styrian Business Promotion Agency (SFG). The Austrian Research Promotion Agency (FFG) has been authorized for the program management. They would furthermore like to express their thanks to Desheng Fu for further input and discussions and Prof. Eugen Brenner and Georg Macher from the Institute of Industrial Informatics of the TU Graz for their support. Also, many thanks to Martin

Krammer from Virtual Vehicle for input and corrections for this work.

References

Bass, Len, Ingo Weber, and Liming Zhu (2015). *DevOps: A software architect's perspective*. Addison-Wesley Professional.

Blochwitz, Torsten et al. (2012). "Functional mockup interface 2.0: The standard for tool independent exchange of simulation models". In: *9th International Modelica Conference*, pp. 173–184. DOI: 10.3384/ecp12076173.

Fernandes, Diogo and Jorge Bernardino (2018). "Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB." In: *DATA*, pp. 373–380.

FMI-Working-Group (2020). *Functional Mock-up Interface for Model Exchange and Co-Simulation*. https://github.com/modelica/fmi-standard/releases/download/v2.0.2/FMI-Specification-2.0.2.pdf. Accessed: 2021-04-12, V 2.0.2.

Hagberg, Aric A., Daniel A. Schult, and Pieter J. Swart (2008). "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, pp. 11–15.

Krammer, Martin and Martin Benedikt (2018). "Configuration of slaves based on the distributed co-simulation protocol". In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. IEEE, pp. 195–202.

Krammer, Martin and Torsten Blochwitz (2018). "The Distributed Co-Simulation Protocol for the Integration of Real-Time Systems and Simulation Environments". In: *Proceedings of the 50th Computer Simulation Conference*. DOI: 10.22360/SummerSim.2018.SCSC.001.

Lanusse, Agnes et al. (2009). "Papyrus UML: an open source toolset for MDA". In: *Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009)*. Citeseer, pp. 1–4.

Modelica Association Project DCP (2019). *DCP Specification Document, Version 1.0*. Linköping, Sweden: Modelica Association. URL: http://www.dcp-standard.org.

Pundsack, Mark (2018). *Out-of-sequence job execution using directed acyclic graphs (DAG) MVC*. https://gitlab.com/gitlab-org/gitlab-foss/issues/47063. Accessed: 2019-11-11.

Reiterer, Stefan and Michael Kalab (2021). "Modelling Deployment Pipelines for Co-Simulations with Graph-Based Metadata". In: *International Journal of Simulation and Process Modelling*. Accepted.

Reiterer, Stefan H. (2020). "Continuous Deployment of ADAS Functions over the Air". In: *13. Grazer Symposium des Virtuellen Fahrzeugs*.

Reiterer, Stefan H. et al. (2020). "Continuous Integration for Vehicle Simulations". In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. IEEE, pp. 1023–1026.

Tick, József (2007). "P-graph-based workflow modelling". In: *Acta Polytechnica Hungarica* 4.1, pp. 75–88.

Portable runtime environments for Python-based FMUs: Adding Docker support to UniFMU

Thomas Schranz[1] Christian Møldrup Legaard[2] Daniella Tola[2] Gerald Schweiger[1]

[1]Graz University of Technology, Austria, {thomas.schranz,gerald.schweiger}@tugraz.at
[2]DIGIT, Department of Electrical and Computer Engineering, Aarhus University, Denmark, {cml,dt}@ece.au.dk

Abstract

Co-simulation is a means to combine and leverage the strengths of different modeling tools, environments and formalisms and has been applied successfully in various domains. The Functional Mock-Up Interface (FMI) is the most commonly used standard for co-simulation. In this paper we extend UniFMU, a tool that allows users to build Functional Mock-Up Units (FMUs) in virtually any programming language, to support execution within Docker. As a result the generated FMUs can be distributed in an environment containing all runtime dependencies. To describe the process of creating Dockerized FMUs using UniFMU, we show how to model and co-simulate a robotic arm and a controller using two Python-based FMUs.

Keywords: FMI, Co-Sim, Python, Tool-Coupling, Docker

1 Introduction

Complex, heterogeneous systems can be found throughout all fields of science and industry. Due to increasing complexity, market competition and specialization, system evaluation and simulation-based analysis has become more and more difficult (G. Schweiger et al. 2019). However, there often exist partial models for different parts of these systems, albeit in different domains and developed using different tools (Gomes et al. 2018). Co-simulation is a means to combine and leverage the strengths of different modeling tools, environments and formalisms (Cremona et al. 2019) and has been applied successfully in various domains (Gerald Schweiger et al. 2018; Pedersen et al. 2017; Nageler et al. 2018). The Functional Mock-Up Interface (FMI) was found to be the most promising standard for continuous time, discrete event, and hybrid co-simulation in a survey by (G. Schweiger et al. 2019). FMI is maintained by the Modelica association (Modelica Association 2021); it can be used to co-simulate components packaged as Functional Mock-Up Units (FMUs), each of which can be built using a different FMI-enabled modeling tool.

1.1 Co-Simulation Tools

With Open Modelica (Asghar and Tariq 2010), Simulink[1] or 20-sim[2] users can generate FMUs based on common modeling languages such as Modelica or MATLAB/Simulink using a graphical interface. The Universal Functional Mock-up Unit (UniFMU) (Legaard et al. 2021) tool allows users to build FMUs from arbitrary code in any programming language; it supports Python, C# and Java out-of-the-box. It uses a precompiled binary wrapper that implements the methods specified in the FMI standard's C-headers to spawn a process that executes the FMU's actual code. This way the FMU can be built from code written in an interpreted language or a language that uses automatic garbage collection. However, this setup, allowing for this kind of flexibility, requires the host machine to provide the process with all runtime dependencies which limits portability, especially between different host machines, and potentially necessitates a complicated setup procedure.

There exists a number of distributed, FMI-based co-simulation tools, many of which were analyzed in (Hatledal et al. 2019). However, all of them require a tight coupling between the co-simulation components and the master algorithm. ProxyFMU, a tool developed by the authors of (Hatledal et al. 2019) decouples the FMUs, in a way that they become independent of the master algorithm in a client/server solution that supports JavaScript, Python, C++ and the JVM on the client side.

The authors of (Hinze et al. 2018) propose a method for running FMUs inside Docker containers by placing the entire FMU archive inside the container and extending the master algorithm with a *remote procedure call* protocol. A distinction between their work and our approach is that FMUs generated using UniFMU work with any FMI-enabled master algorithm without the need to implement any additional protocols.

1.2 Contributions

In this paper we extend UniFMU using the virtualization environment Docker[3], such that the FMUs can be shipped with all runtime dependencies. We provide a general

[1] http://www.mathworks.com/products/simulink
[2] http://www.20sim.com
[3] https://www.docker.com

DOI
10.3384/ecp21181419

mechanism that can be leveraged for all languages supported by the tool. The resulting FMUs have nearly the same portability as compiled FMUs (except for the dependency on Docker) and require no language-specific setup procedure, but still allow the use of non-compiled languages and languages that use automatic garbage collection. To explain the process of creating Dockerized FMUs using UniFMU, we show how to model and co-simulate a robotic arm and a controller using two Python-based FMUs. The UniFMU tool (with the extensions for Dockerization) is available on Github[4]. The FMUs described in this paper can be found in a separate repository[5].

The rest of the paper is structured as follows. First, section 2 introduces a robotic arm and a controller which is used as a case study throughout the paper. Next, section 3 provides an introduction to UniFMU and describe how the robotic arm and controller can be implemented as FMUs using the tool. Then, section 4 describes the extension of UniFMU that allows FMUs and their dependencies to be deployed inside Docker containers. Afterwards, section 6 provides a discussion of the results and outlines future work on the tool. Finally, section 7 provides concluding remarks.

2 Case Study

To exemplify the process of using UniFMU we consider the case of modeling a robotic arm coupled to a controller as depicted in Figure 1. The example is chosen to highlight how different modeling formalisms can be realized by the tool. Specifically, the robotic arm described in subsection 2.1 is inherently continuous, whereas the controller described in subsection 2.2 is discrete.

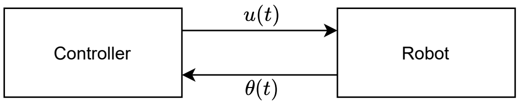


Figure 1. Connection between controller and robot model.

2.1 Robotic Arm

The robotic arm is modeled as a controlled inverted pendulum. The states of the system are its angle θ, the angular velocity ω and the current running through the coils of the electrical motor i. The dynamics of the robotic arm are described by Equation 1. Note that contrary to the visualization shown in Figure 7 the model only considers a single joint that rotates around a single axis.

$$f(x) = \begin{bmatrix} \dot{\theta} \\ \dot{\omega} \\ \dot{i} \end{bmatrix} = \begin{bmatrix} \omega \\ \frac{K \cdot i - b \cdot \omega - m \cdot g \cdot l \cdot cos(\theta)}{J} \\ \frac{u \cdot V_{abs} - R \cdot i - K \cdot \omega}{L} \end{bmatrix} \tag{1}$$

where:

The derivative of the angle $\dot{\theta}$ is, per definition, equal to the velocity of the arm ω. The derivative of the angular velocity $\dot{\omega}$ is determined by the torque coefficient $K = 7.45\ s^{-2}A^{-1}$, the current i, the motor-shaft friction $b = 5.0\ kg \cdot m^2 \cdot s^{-1}$ and the gravity acting on the arm, denoted by $m \cdot g \cdot l \cdot \cos(\theta)$, with $m = 5.0\ kg$, $g = 9.81ms^{-2}, l = 1.0\ m$. The change in current is determined by the input from the controller u, the voltage across the coils $V_{abs} = 12.0\ V$, the resistance $R = 0.15\ \Omega$ and the motor's inductance $L = 0.036\ H$.

2.2 Controller

A proportional-integral-derivative (PID) controller (Åström and Murray 2010) is used to generate the control signals sent to the robotic arm. The continuous formalization of the controller is given by:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \dot{e}(t) \tag{2}$$

where $e(t)$ is a measure of the error of the variable being controlled and K_p, K_i and K_d are coefficients used to tune how the proportional and derivative terms are weighted. In case of the robot, the controller is trying to minimize the error between the desired angle $\theta^*(t)$ and the true angle $\theta(t)$. Thus, the error is defined as $e(t) = \theta^*(t) - \theta(t)$.

In practice, most controllers are implemented digitally, which means that derivatives and integrals must be replaced by discrete approximations. There are several ways to do this, the simplest being to replace derivatives by first-order differences

$$\dot{e}(t_k) \approx \dot{e}_k = \frac{e_k - e_{k-1}}{T},$$

and integrals by sums

$$\int_0^{t_k} e(t_k) \approx E_k = \sum_{n=1}^{N} e_{k_n} \cdot T$$

where $e_k = e(t_k)$, T is the sampling time and $N = t_k/T$ is the number of samples between time 0 and t_k. After replacing the continuous definitions in Equation 2 we obtain an equation that can be implemented on a discrete controller

$$u_k = K_p e_k + K_i E_k + K_d \dot{e}_k \tag{3}$$

This discretization scheme is simple to implement

3 Modeling

In this section, we describe how UniFMU is installed and how it is used to generate an FMU. We provide a brief overview of the resulting FMU's structure and method of operation. Subsequently, we describe the FMUs used to model the robotic arm and the controller. For illustrative purposes both FMUs are implemented in Python, however in the general case they can be implemented in a mix of languages.

[4] `https://github.com/INTO-CPS-Association/unifmu`

[5] `https://github.com/Daniella1/robot_unifmu`

DOI 10.3384/ecp21181419

3.1 Creating an FMU using UniFMU

UniFMU is a command line interface (CLI) that can be installed through Python's package installer `pip` or from source following the instructions in the official repository. Installation through `pip` uses a single command:

```
pip install unifmu
```

It should be noted that the FMUs generated with UniFMU do not require Python during runtime, unless the FMUs themselves are implemented in Python. To generate an FMU the tool has to be invoked with the subcommand `generate`, and supplied with the language the FMU is implemented in and the name it should have; for a Python-based FMU with the name `robot` this looks like:

```
unifmu generate python robot
```

This generates an FMU with the structure shown in Figure 2. The `binaries` directory contains a precompiled wrapper for Windows, Linux and MacOS that implements the methods specified in the FMI's C-headers and relays them to the actual implementation of the FMU found in the `resources` directory. `model.py` defines a class that declares a set of methods that correspond to the methods in the FMI standard, such as FMI's `fmi2DoStep` which is implemented by the Python method `do_step`. The actual overwrite used to model the robotic arm can be seen in Listing 1.

```
robot.fmu
├── binaries
│   ├── darwin64
│   ├── linux64
│   │   └── unifmu.so
│   └── win64
├── resources
│   └── model.py
└── modelDescription.xml
```

Figure 2. The directory structure for a Python FMU. Note that several files generated by the tool are omitted for simplicity.

3.2 Robotic Arm FMU

The robotic arm FMU is implemented in Python using a numerical solver provided by the *SciPy* (Virtanen et al. 2020) package. The general procedure for solving an ODE using Scipy is to define a function which evaluates the derivative for a given combination of state and time. Using Equation 1 as a reference the function $f(\cdot)$ can be defined as shown in Listing 1.

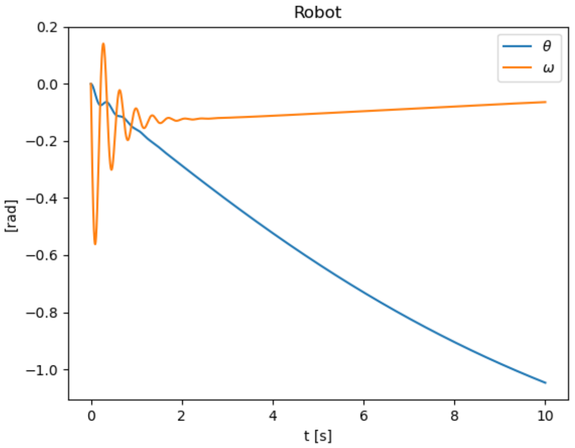


Figure 3. Standalone test of robotic arm, with values $\theta_0 = \omega_0 = i_0 = u(t) = 0$.

```
def do_step(self,current_time,step_size,no_step_prior):
    def f(t, y):
        theta, omega, i = y
        tau=self.k1*np.cos(theta)
        domega=(self.K*i-self.b*omega-tau)/self.J
        di=(self.V-self.R*i-self.K*omega)/self.L
        dtheta=omega
        return dtheta, domega, di
    res=solve_ivp(f,(current_time,current_time+step_size),y0)
    self.theta,self.omega,self.i=res.y[:,-1]
    return Fmi2Status.ok
```

Listing 1. Implementation of the `fmi2DoStep` method for the robotic arm FMU.

Given the definition of the derivative, the `solve_ivp` function can be used to obtain the solution for the next step of the FMU and allows users to choose between solvers. However, it is also possible to use any other Python library providing numerical solvers or to implement a custom solver. This is a very flexible solution as it allows users to choose the type of solver that is suitable for the particular ODE. After solving the ODE, the newly estimated state is assigned to the instance, where it can be accessed from other methods and the FMI.

To test the dynamics of the robotic arm FMU, a small test program is written in Python which invokes the *do_step* several times. The results are shown in Figure 3 for initial state and input $\theta_0 = \omega_0 = i_0 = u(t) = 0$. We see that the angle of the robot decreases from 0 to -1 over 10 seconds.

3.3 Controller FMU

The controller-FMU implements a simple control algorithm that determines the signal sent to the motor based on the difference between the desired and the actual current angle. Similarly to how Equation 1 was translated into a

Python function describing the derivative of the state, we use the control policy Equation 3 as a reference to implement the expression shown in Listing 2.

```python
def do_step(self, current_time, step_size, no_step_prior):
    err=self.setpoint_t-self.measured_t
    self.I=self.I+err*step_size
    D=(err-self.p_err)/step_size
    self.u=self.Kp*err+self.Ki*self.I+self.Kd*D
    self.p_err = err
    return Fmi2Status.ok
```

Listing 2. Implementation of the `fmi2DoStep` method for the controller-FMU.

In most practical situations, controllers are implemented on a processing unit where updates to the output would happen at a fixed update rate determined by the controller's clock frequency and the number of operations needed at each update.

An implicit assumption of our model is that the step-size used by the solver matches the update rate of the controller. For small step sizes, the discrete approximation implemented by the model remains relatively accurate. However, for larger step sizes the accuracy of the discretization scheme is reduced, which may ultimately cause the closed-loop system to become unstable. A solution to mitigate the discretization error is to use a more sophisticated discretization scheme, such as Tustin's method (Franklin, Powell, and Emami-Naeini 2020).

As in the robotic arm FMU, we can also use any external library for modeling the controller. For instance, a package such as *python-control*[6], can be used to evaluate the performance of different controllers.

The functionality of the controller can be verified by writing a small test program in Python that invokes the *do_step* method of the FMU. To examine the closed-loop behavior, the robot is replaced with a simple linear model described by the ODE $\dot{\theta} = u$. Executing the Python test program, we obtain the step-response of the closed-loop system (with surrogate model) as depicted in Figure 4.

4 Docker Support

A key contribution of this paper is extending UniFMU so that the generated FMUs can be executed within a virtualization environment using Docker. To create a Dockerized FMU the user can append the *–dockerize* switch to UniFMU's `generate` subcommand:

```
unifmu generate python --dockerize robot
```

The functionality is available for Linux and macOS and all languages that the tool supports. Windows support is under development, but is held back by limitations of Docker's networking capabilities when running on Windows.

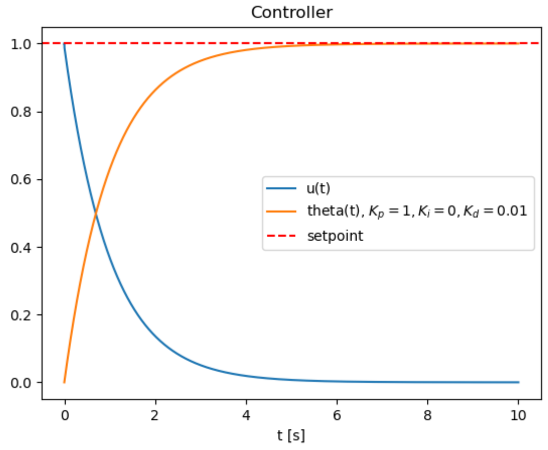


Figure 4. Standalone test of the controller FMU with using linear model for plant $\dot{\theta} = u$, step size = 0.001, setpoint = 1.0

4.1 Setting up the image

A configuration file, referred to as the `Dockerfile`, provides instructions to build the environment on any host machine. An excerpt from the `Dockerfile` used by the robotic arm FMU can be seen in Listing 3. The first line declares that the image for the FMU is assembled ontop a pre-built Python 3.8. image from the Docker container library. The second line invokes the package manager `pip` to install packages required by the model. For simplicity, the three dots represent the dependencies required by the Python backend to communicate with the binary. The third line instructs Docker to copy the `container_bundle` directory into the image. The `container_bundle` contains all files that are needed during runtime, such as the actual model implementation and all user-generated files and dependencies.

```dockerfile
FROM python:3.8
RUN pip install ... scipy roboticstoolbox-python matplotlib
COPY container_bundle resources
...
```

Listing 3. Dockerfile used to assemble the image used by FMU instances.

4.2 Instantiating a Dockerized FMU

The process of creating an instance of a Dockerized FMU is depicted in Figure 5. The steps are as follows: First, the binary will ensure that the image declared in the Dockerfile has been built. If this is not the case, it will automatically invoke the `Dockerfile` to build the image. Next, from the image a container is created. The container has access to all dependencies listed in the `Dockerfile`, such as Python packages that were installed through `pip` and everything inside the `container_bundle`. Note that each instance of an FMU is executed within its own container and removed after use. This ensures that no instances of an FMU share any state or influence each other directly.

[6] `https://python-control.readthedocs.io/en/0.8.3/index.html`

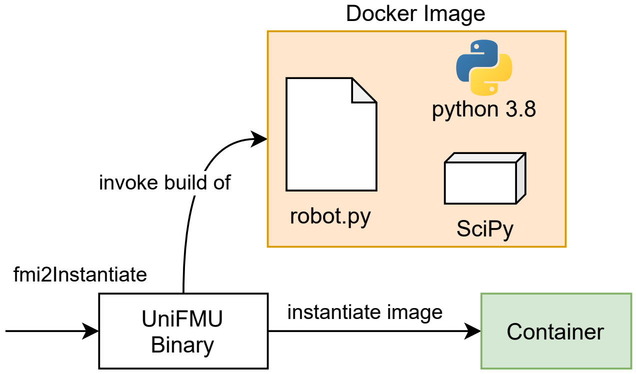


Figure 5. Deployment of a model inside the Docker container.

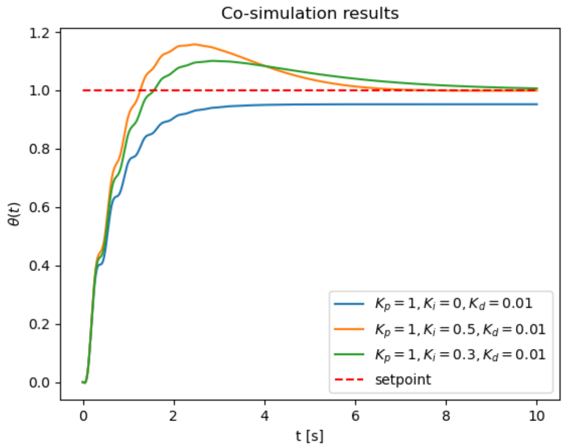


Figure 6. Co-simulation of the controller and the robot with the $setpoint = 1$. Experiments of varying the controller parameters are shown.

5 Results

A co-simulation was configured and run using the two FMUs with the INTO-CPS tool-chain (Larsen et al. 2016). We used a fixed step-size solver with a step-size of 0.001 seconds, set the desired angle to 1 radian and plotted θ as a function of time for various values of K_p, K_i and K_d. The corresponding plot can be seen in Figure 6. Fig. 7 shows the robotic arm at an angle of 1 radian. The controller with $K_i = 0$ exhibits a substantial steady-state error, whereas the ones with an integral term converge within 10 seconds. Besides, it can be seen that controllers with an integral term cause the system to overshoot the setpoint. Tweaking the coefficients of the controller allows us to balance the tendency to overshoot and the steady-state error, such that they meet the requirements of the application. Methods based on heuristics exist for tuning PID controllers, which could be applied to tune the controller for the robotic arm. However, we considered applying these to be beyond the scope of this example use case.

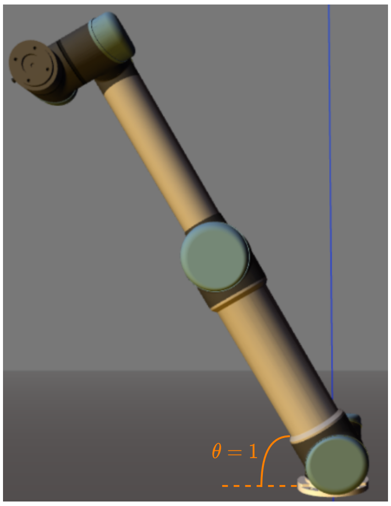


Figure 7. A visualization of the robot during the co-simulation. The measured θ is equal to the $setpoint = 1$ rad, when controlled with the PID-controller.

6 Discussion

A central objective of the FMI standard is to facilitate the exchange of models generated by different tools. To do so, FMI requires communication through a C-API, which complicates implementing models in languages that cannot be compiled into a C-compatible binary. UniFMU circumvents this issue by providing a generic C-binary that handles all communication between FMI calls to the FMU and the FMU's actual code. Being able to use high-level programming languages such as Python allows developers to leverage a large ecosystem of scientific libraries and thus implement models quickly and efficiently, especially in contrast to writing everything from scratch. Consider the implementation of the `do_step` method for the robotic arm shown in Listing 1. The ODE is declared and solved in eight lines of code. We believe that this has the potential to simplify co-simulation for more modeling applications and engage more developers.

Another aspect to this approach is that the resulting FMUs can be verified and debugged using the development tools of the FMU's language. For instance, it allowed us to write small test programs for verifying the FMUs before performing the co-simulation of the system. In our experience, the ability to effectively test the individual models greatly reduces the number of issues encountered when integrating the models.

Using FMUs that require runtime dependencies to be handled manually is counter-intuitive to the idea of simple, standardized model exchange. Consequently, in this work we addressed this issue by providing a way to automatically virtualize the runtime environment with all dependencies inside a Docker container rather than requiring the host machine to provide a suitable environment. The way this Dockerization was implemented did not affect UniFMU's precompiled binaries and all changes to the language-specific backends are simply additional configuration options instead of hard dependencies on Docker itself. The latter might be reused in the future to implement remote deployment.

7 Conclusion

Co-simulation is a key research interest. The FMI standard is among the most popular interfaces for model exchange and co-simulation. There are various tools to generate FMI-compliant FMUs. UniFMU is one such tool that allows users to build FMUs from arbitrary code written in any language. We used UniFMU to generate two Python-based FMUs in order to co-simulate a robotic arm and a controller. However, the resulting FMUs required the host machine to provide a Python runtime environment with all dependencies preinstalled, effectively limiting portability and ease of deployment. To address this issue we extended UniFMU using the virtualization tool Docker. With our extension, UniFMU is able to generate FMUs shipped with a `Dockerfile` that automatically builds a runtime environment inside a container. This way the FMUs are almost as portable as compiled FMUs (except for the dependency on Docker) but still support the use of non-compiled languages. Our extension is not limited to Python but can be reused for other languages as well. Besides, the changes we implemented can help in developing a configuration for remote deployment of FMUs in the future.

Acknowledgements

The authors would like to thank Thomas Schwengler for their support with designing the Docker-integration.

The reported research was conducted within the project NextHyb2 (881150) and project DigitalEnergyTwin (873599), which received funding in the framework of ”Stadt der Zukunft” and "Energieforschung", a research and technology program of the Austrian Ministry for Transport, Innovation and Technology (BMVIT).

References

Asghar, Syed Adeel and Sonia Tariq (2010). *Design and Implementation of a User Friendly OpenModelica Graphical Connection Editor*. eng.

Åström, Karl Johan and Richard M Murray (2010). *Feedback Systems: An Introduction for Scientists and Engineers*. In English. ISBN: 978-1-4008-2873-9.

Cremona, Fabio et al. (2019-06). “Hybrid Co-Simulation: It’s about Time”. en. In: *Software & Systems Modeling* 18.3, pp. 1655–1679. ISSN: 1619-1366, 1619-1374. DOI: 10.1007/s10270-017-0633-6.

Franklin, Gene F., J. David Powell, and Abbas Emami-Naeini (2020). *Feedback Control of Dynamic Systems*. eng. Eighth edition, global edition. Harlow, United Kingdom: Pearson Education Limited. ISBN: 978-1-292-27452-2.

Gomes, Cláudio et al. (2018-07). “Co-Simulation: A Survey”. en. In: *ACM Computing Surveys* 51.3, pp. 1–33. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3179993.

Hatledal, Lars Ivar et al. (2019-02). “FMU-proxy: A Framework for Distributed Access to Functional Mock-up Units”. In: pp. 79–86. DOI: 10.3384/ecp1915779. URL: https://ep.liu.se/en/conference-article.aspx?series=ecp&issue=157&Article_No=8 (visited on 2021-05-09).

Hinze, Christoph et al. (2018). “Towards Real-Time Capable Simulations with a Containerized Simulation Environment”. In: *2018 25th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, pp. 1–6. DOI: 10.1109/M2VIP.2018.8600827.

Larsen, Peter Gorm et al. (2016). “Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project”. In: *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*, pp. 1–6. DOI: 10.1109/CPSData.2016.7496424.

Legaard, Christian Møldrup et al. (2021). “A Universal Mechanism for Implementing Functional Mock-up Units”. In: *11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. SIMULTECH 2021. Virtual Event, to appear.

Modelica Association (2021). *Functional Mock-up Interface for Model Exchange and Co-Simulation*. https://www.fmi-standard.org/downloads.

Nageler, P. et al. (2018-08). “Novel method to simulate large-scale thermal city models”. en. In: *Energy* 157, pp. 633–646. ISSN: 03605442. DOI: 10.1016/j.energy.2018.05.190. URL: https://linkinghub.elsevier.com/retrieve/pii/S0360544218310363 (visited on 2021-05-06).

Pedersen, Nicolai et al. (2017). “Distributed Co-Simulation of Embedded Control Software with Exhaust Gas Recirculation Water Handling System using INTO-CPS:” in: *Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. Madrid, Spain: SCITEPRESS - Science and Technology Publications, pp. 73–82. ISBN: 9789897582653. DOI: 10.5220/0006412700730082. URL: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006412700730082 (visited on 2021-05-06).

Schweiger, G. et al. (2019-09). “An empirical survey on co-simulation: Promising standards, challenges and research needs”. en. In: *Simulation Modelling Practice and Theory* 95, pp. 148–163. ISSN: 1569190X. DOI: 10.1016/j.simpat.2019.05.001. URL: https://linkinghub.elsevier.com/retrieve/pii/S1569190X1930053X (visited on 2021-05-06).

Schweiger, Gerald et al. (2018-12). “District energy systems: Modelling paradigms and general-purpose tools”. en. In: *Energy* 164, pp. 1326–1340. ISSN: 03605442. DOI: 10.1016/j.energy.2018.08.193. URL: https://linkinghub.elsevier.com/retrieve/pii/S0360544218317274 (visited on 2021-05-06).

Virtanen, Pauli et al. (2020). “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17, pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

General Purpose Lua Interpreter for Modelica

Fabian Buse[1] Tobias Bellmann[1]

[1]Institute of System Dynamics and Control, German Aerospace Center (DLR), Germany,
{fabian.buse,tobias.bellmann}@dlr.de

Abstract

Simulation becomes more and more important in the development of complex systems. Modeled systems often are comprised of mechanical, electrical as well as software systems. It is often not possible to evaluate the performance of a system without considering some higher level logic anymore. Scripting languages, such as Lua, are usually well suited to implement these logic elements. This paper shows the integration of the Lua interpreter into Modelica, and gives examples how the library can be used to help with the simulation of industrial robots or in the development of a planetary exploration rover in the MMX Mission.

Keywords: Modelica, Lua, script language, robotics, finite state machine

1 Introduction

When simulating complex systems, it is often necessary to model not only the mechanics, electronics and control system correctly, but also the higher-level logic that governs the general behavior. However, this high-level logic is generally difficult to implement with native Modelica methods. Although it is possible to build a control logic based on multiple inputs and outputs, e.g. by using state machines, it is error-prone and not very flexible. The requirement to balance all equations and unknowns can be difficult and tedious if the different control states or modes have distinct structures. If a model is to be used to test and develop this high-level logic, the limitations of native Modelica methods are even more pronounced. The developer with in-depth Modelica knowledge may not be responsible for developing and testing the high-level logic. One common method to solve this problem is the FMI standard (Blockwitz et al. 2012). With this approach the system model is built in Modelica and then imported into a different tool that is better suited to handle high-level logic. The Modelica Lua library presented in this paper was developed to combine Modelica's strengths in physical modeling with the flexibility of a script-based interpreter to solve the problem stated before without the need of an additional tool.

The concept of this library is an extension and generalization of the Lua interpreter presented and developed for the DLR Robots library (Bellmann, Seefried, and Thiele 2020). Lua was chosen due to its lightweight, well maintained interpreter, its simple and easy to learn syntax and already established application in other fields, such as game development. With Lua being a commonly used language, many tools and third party libraries are already available to use.

1.1 Design Goal

The goal for this library was to provide an application independent Lua interpreter with a lightweight interface. The integration into existing Modelica models should be as easy as possible. All basic Modelica data types, Real, Integer, Boolean and String, should be supported as input and output of Lua. In order to meet the requirements of most applications, different modes of operation shall be supported, both a loop-like repetition of the Lua script and a single and serial execution of the script. To allow multiple instances or different systems controlled by Lua in a single Modelica model, it is desired to allow multiple, independent instances of the Lua interpreter in the same Modelica model. It should also be possible to write Lua in Modelica directly as a string or to load a script from disk. It should be possible to use the full Lua functions and also to enable third-party Lua libraries. Finally, the Lua library should be platform-independent.

2 Structure of the library

From the previously stated goals a simple structure of the library has been developed. The following section will explain the general setup, both interfaces to the Modelica and Lua side of the library as well as some further considerations.

2.1 Implementation

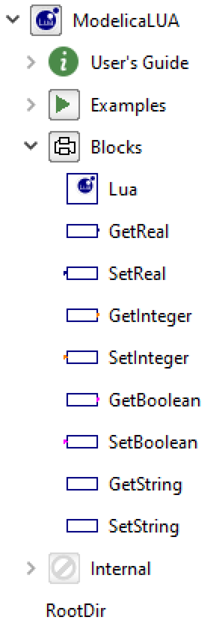
Similar to the implementation of the Lua interpreter in the DLR Robots library, each Lua interpreter instance runs in a separate thread independent of the main Modelica thread. The data exchange between the interpreter and Modelica takes place via a data core which can be accessed by Modelica as well as Lua. Extending from the implementation in the DLR Robots library, where no further synchronization between Lua and Modelica was possible, the new Lua library offers various options. Similar to the original implementation, both programs can run completely independently of each other, with the timing on both sides being handled manually by custom signals transmitted through the data core. To enable the timed execution of certain instructions, waiting functions based either on simulation-time or CPU-time are now provided.

DOI
10.3384/ecp21181425

Alternatively, the Lua code can be synchronized to specific time events. This can be further configured so that Lua waits for Modelica, Modelica for Lua, or both. If synchronization is activated, the data is only read and written to the data core at this time to ensure data coherence.

2.2 Modelica Interface

The Modelica library, shown in Figure 1, was kept as simple as possible. The communication, script selection and synchronization is handled by the main Lua block, inputs and outputs are handled by their type specific blocks. To reduce the total number of required blocks all interfaces except strings are treated as vectors.



- ModelicaLUA
  - User's Guide
  - Examples
  - Blocks
    - Lua
    - GetReal
    - SetReal
    - GetInteger
    - SetInteger
    - GetBoolean
    - SetBoolean
    - GetString
    - SetString
  - Internal
  - RootDir

Figure 1. Structure of the Modelica Lua library as shown in Dymola.

To define the hierarchy and enable multiple interpreter instances an inner outer construct was chosen. Each instance of a `Lua` block holds one independent interpreter, thus the set and get blocks have their interpreter assigned based on the model structure in Modelica. The main `Lua` block contains all parameters defining the created instance. These parameters can be divided into three categories, script, timing and synchronization. The script category, defines what script to load or if selected, the string containing the entire script, as well as a unique name identifying the interpreter instance. Additionally, further search directories for locating scripts or libraries can be defined. The timing category defines the sampling time for communication, as well as the start time for the script. The different synchronization modes are configured here as well, if synchronization is enabled, the communication time step is used as the synchronization time step. In contrast, each get or set block is only parameterized by its name and dimension. If the dimension of a value in the data core does not match the requested size, the returned data will be truncated or padded with zeros.

One critical feature for a flexible usage of this library is a robust initialization setup. To ensure that the representation in the data core is always consistent with the model state in Modelica, the initialization of both get and set blocks is handled in Modelica. For the set functions the initial value of the input is simply communicated in an initial equation to the data core. For the get blocks either an explicit start value can be defined or the start value can be implicitly defined by the Modelica models and equations connected to its output.

2.3 Lua Interpreter

The Lua interpreter itself is based on the official sources provided by the Lua community[1]. By encapsulating both the Lua stack and an additional data broker into an external object, each instance of the `Lua` block has separate interpreters. Due to the separation of the Modelica and Lua thread an intermediate data core is required. On both sides bindings to the data core have been implemented to enable thread safe access. The corresponding Modelica blocks were shown above. Lua bindings are provided in an additional Lua library `modelica.lua`, see Table 1, its functions correspond to the Modelica get and set blocks as well as additional Modelica interfaces. Get methods in Lua have an optional `dim` parameter. If `dim` is defined and larger than zero, the returned data will have this dimension and will be truncated or padded with zeros to match the desired size whereas if it is not defined or zero, the returned value has the dimension currently present in the data core.

2.4 Synchronization

The functions `sync`, `wait` and `wait_until` provide means to synchronize the execution of the Lua code to Modelica. Two approaches are available. First, a program is executed sequentially and uses the `wait` and `wait_until` to perform actions at predefined times. Both functions block until the required time has passed:

```lua
local t = 0
t = Modelica.time() -- t = 0.0
Modelica.wait_until(10)
t = Modelica.time() -- t = 10.0
Modelica.wait(2)
t = Modelica.time() -- t = 12.0
```

The `wait` function can either use simulation or CPU time, with simulation time as a default. Whereas the `wait_until` is always linked to simulation time. Alternatively, the code in Lua can be structured in loop, where every cycle is synchronized to a sampled clock in Modelica by using `sync`:

```lua
local t = 0
while(Modelica.sync())
do
  -- this loop is executed with
  -- the sample time defined in Modelica
  t = Modelica.time()
  -- t = current simulation time
end
```

The `sync` function blocks the Lua thread until the next pulse of the clock in Modelica, its rate is defined in the parameters of the `Lua` block. If the execution of the Lua code takes longer than the defined sample rate, it is possible to block the execution in the Modelica thread until the `sync` function is reached again.

2.5 Lua Libraries

One of the key features of Lua is the ability to use additional libraries. This enables not only the usage of the extensive Lua

[1] `https://www.lua.org/`

Table 1. Modelica interface functions provide in Lua.

Function	*Description*
`sync()`	synchronizes Lua to Modelica
`dt()`	returns the current Lua time step, only works if `sync` is used
`time()`	returns the current simulation time in seconds
`setReal(name,u)`	writes a real value or vector `u` to the data core
`setInteger(name,u)`	writes an integer value or vector `u` to the data core
`setBoolean(name,u)`	writes a boolean value or vector `u` to the data core
`setString(name,u)`	writes a single string `u` to the data core
`getReal(name,dim)`	reads a real value or vector from the data core, `dim` is optional
`getInteger(name,dim)`	returns an integer value or vector from the data core, `dim` is optional
`getBoolean(name,dim)`	returns a boolean value or vector from the data core, `dim` is optional
`getString(name)`	returns a single string from the data core
`wait(duration, cpu_time)`	blocks for a specific duration, either in simulation or cpu time, if `cpu_time` is not set it defaults to `false`
`wait_until(time)`	blocks until the simulation time is reached
`print(msg)`	prints a message with the `ModelicaFormatMessage` function, usually to the log file resulting from model execution
`terminate(msg)`	terminates the simulation, equivalent to `terminate` in Modelica

standard library which already supports basic file system support, basic math functions, and string manipulations but also useful features like 3D vector math (Bjorn 2021) or finite state machines (Conroy 2021). These libraries can be easily added to the Lua script via the Lua `require` functionality, which is similar to the C/C++ include. This `require` functionality is also used to include the Modelica add-on to each loaded Lua script.

2.6 Logging

During the execution of a Lua script, the user can print out messages over the logging interface. Several output channels can be used:

- Log messages to a file:

  `Logger.createFileOutput(title, filename,append)`

- The system console is used for logging:

  `Logger.createConsoleOutput(title)`

- The Modelica message command is used to log to the Modelica tool:

  `Logger.createModelicaLogOutput(title)`

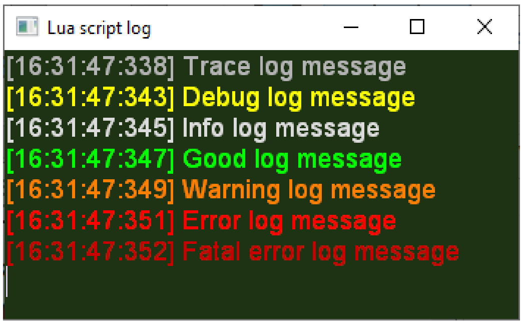
- A logging window is created to log messages in. (MS Windows only):

  `Logger.createLogWindow(title,x,y,w,h)`

After the initialization, the `Logger.log` command can be used to log information in one of the output channels. The channel to be used is defined by the parameter string `title`:

```
Logger.createLogWindow('Lua script log'
    ,100,100,300,150);
Logger.log('Lua script log', 'Some text
    for the console')
```

It is also possible to add a timestamp at the beginning of the text or, in case of the log window define different colors via the severity flag (See Figure 2) or filter the shown messages (errors only, errors and warnings, all) by setting the verbosity of the log channel.



```
[16:31:47:338] Trace log message
[16:31:47:343] Debug log message
[16:31:47:345] Info log message
[16:31:47:347] Good log message
[16:31:47:349] Warning log message
[16:31:47:351] Error log message
[16:31:47:352] Fatal error log message
```

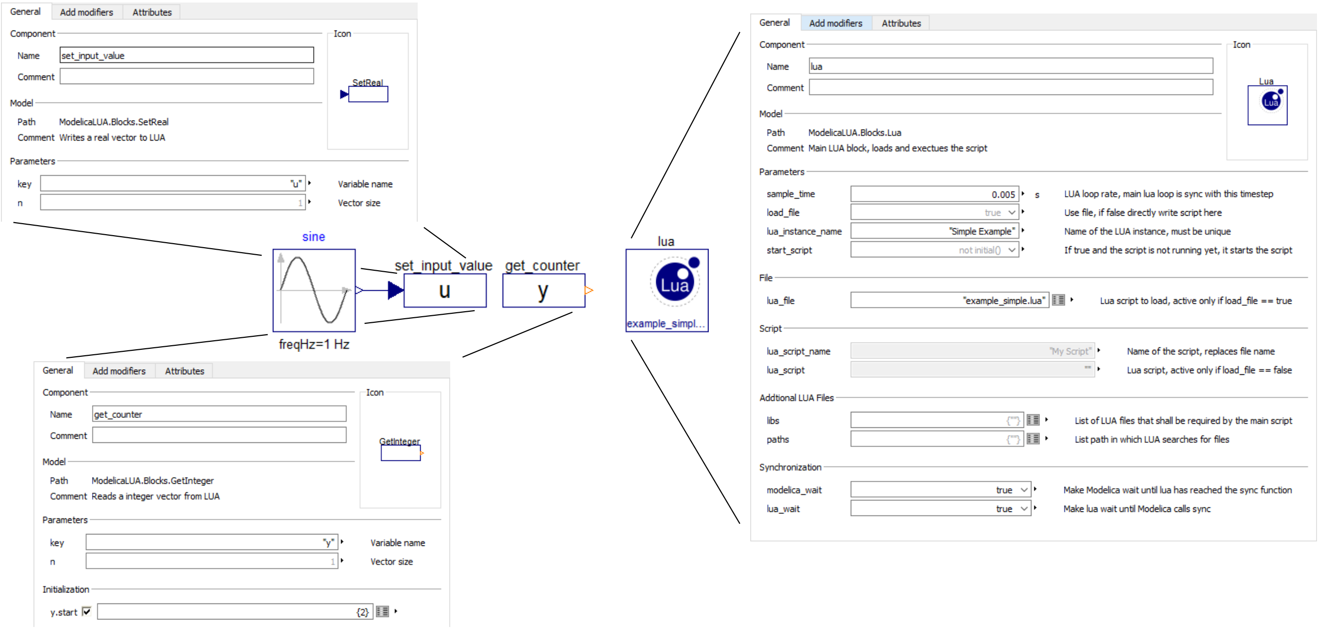
Figure 2. Logging window with log messages of varying severity and timestamps.

3 Examples

3.1 Library Examples

To provide an overview and show the functionalities some of the library's simple examples models will be shown here. The examples in this section would be rather easy to replicate in pure Modelica but are designed to highlight some of the core features.

The first example, shown in Figure 3, uses a synchronized Lua script to count the number of times an input value `u` exceeds a threshold of 0.5 and output it as `y`. Once the counter exceeds five, it is reset to zero. The initial value of the counter is set in Modelica. The used Lua code:



Component
Name: set_input_value
Comment:
Model
Path: ModelicaLUA.Blocks.SetReal
Comment: Writes a real vector to LUA
Parameters
key: "u" — Variable name
n: 1 — Vector size

Component
Name: lua
Comment:
Model
Path: ModelicaLUA.Blocks.Lua
Comment: Main LUA block, loads and exectues the script
Parameters
sample_time: 0.005 s — LUA loop rate, main lua loop is sync with this timestep
load_file: true — Use file, if false directly write script here
lua_instance_name: "Simple Example" — Name of the LUA instance, must be unique
start_script: not initial() — If true and the script is not running yet, it starts the script
File
lua_file: "example_simple.lua" — Lua script to load, active only if load_file == true
Script
lua_script_name: "My Script" — Name of the script, replaces file name
lua_script: "" — Lua script, active only if load_file == false
Addtional LUA Files
libs: {""} — List of LUA files that shall be required by the main script
paths: {""} — List path in which LUA searches for files
Synchronization
modelica_wait: true — Make Modelica wait until lua has reached the sync function
lua_wait: true — Make lua wait until Modelica calls sync

Component
Name: get_counter
Comment:
Model
Path: ModelicaLUA.Blocks.GetInteger
Comment: Reads a integer vector from LUA
Parameters
key: "y" — Variable name
n: 1 — Vector size
Initialization
y.start ☑ {2}

Figure 3. Diagram layer of a simple Lua example. Parameters of the Lua, input and output blocks are shown in more details.

```
local hysteresis = true
local threshold = 0.5
while(Modelica.sync())
do
 local u = Modelica.getReal('u')
 local y = Modelica.getInteger('y')
 if u > threshold and hysteresis then
  y = y + 1
  hysteresis = false
 elseif u < threshold then
  hysteresis = true
 end
 if y > 5 then
  y = 0
 end
 Modelica.setInteger('y',y)
end
```

reads out the initial values of the counter an then goes into a while loop that is synchronized to Modelica with the `sync` function. A simple **`if elseif`** block increments the counter `y` when the input `u` exceeds the threshold. The resulting behavior is shown in Figure 4.

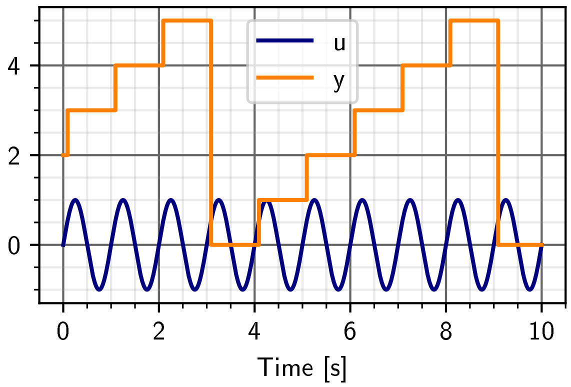


Figure 4. Input `u` and output `y` of the simple Lua example.

A second example uses the publicly available finite state machine implementation by (Conroy 2021). In this example a state machine switches between its states `Green`, `Yellow`, `Red` and `Black` based on thresholds of a single input signal. When the state `Red` is entered the counter `nAlerts` is incremented by one, once this counter exceeds 20 the final state `Black` is activated. The behavior of the output variable `result` can be defined for each state individually. In case of the state `Red`, `result` variable will hold the time since the state became active. A variable `state` is used to communicate the active state encoded into an integer to Modelica. The Modelica model, shown in Figure 5 is rather simple and just provides the necessary inputs and outputs. The Lua code first defines the structure of the state machine based on its state transitions by defining the event name as well as the start and end of the transition. Optionally `onEnter`, `onExit` and `run` functions can be defined for each sate. This structure and the function definition for the `Red` state, as well as the Lua code triggering the events is shown here:

```
-- import state machine library
local machine = require('statemachine')
local nAlerts = 0
local tRed = 0 -- time when red becomes
    active
-- state machine definition
local fsm = machine.create({
initial = 'Green',
events = { -- event definitions
 { name = 'warn',
   from = 'Green',
   to = 'Yellow' },
 { name = 'alarm',
   from = 'Yellow',
   to = 'Red'     },
 { name = 'calm',
   from = 'Red',
```

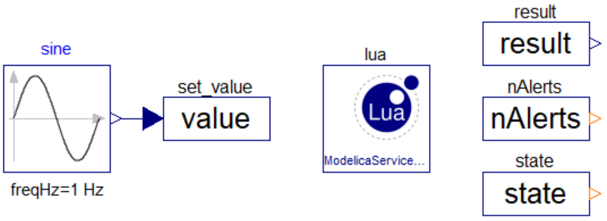


Figure 5. Input `value` and outputs `result`, `nAlerts` and `state` of the simple state machine Lua example

```
   to = 'Yellow' },
 { name = 'clear',
   from = {'Yellow','Red' },
   to = 'Green'  },
 { name = 'panic',
   from = 'Red',
   to = 'Black'},
},

callbacks = { -- callback definitions

-- define function when Red becomes
    active
onenterRed= function()
 Modelica.print('Entering Red')
 nAlerts = nAlerts + 1
 Modelica.setInteger('nAlerts',nAlerts)
 Modelica.setInteger('state',3)
 tRed = Modelica.time()
end,

-- define function when Red becomes
    inactive
onleaveRed = function()
 Modelica.print('Leaving Red')
end,

-- define function while Red is active
runRed = function()
 result = Modelica.time() - tRed
end,

-- callback definitions for other states
     skipped

}})
-- state machine fully defined
-- main loop, executed in sync with
    Modelica
while(Modelica.sync())
do
 local value = Modelica.getReal('value')
 -- trigger events
 if nAlerts > 20 then
  fsm:panic()
 elseif value > 0.9 then
  fsm:alarm()
 elseif value > 0.5 then
  fsm:warn()
 elseif value < 0.0 then
  fsm:clear()
 elseif value < 0.5 then
  fsm:calm()
 end
 fsm:run() -- executes run of active
     state
 Modelica.setReal('result',result)
end
```

In this case the state transitions are triggered by different thresholds of the input `value`. The `fsm:run()` call executes the `run` function associated with the active state. Since the `Modelica.print()` function is called in every states' `onenter` and `onleave` functions, the state-machine behavior can be observed in the log generated by Modelica:

```
Lua[Example] @0.085: Leaving Green
Lua[Example] @0.09: Entering Yellow
Lua[Example] @0.18: Leaving Yellow
Lua[Example] @0.185: Entering Red
Lua[Example] @0.42: Leaving Red
Lua[Example] @0.425: Entering Yellow
 ...
Lua[Example] @20.18: Leaving Yellow
Lua[Example] @20.185: Entering Red
Lua[Example] @20.185: Leaving Red
Lua[Example] @20.19: Entering Black
```

Alternatively the previously described logger could be used in a similar manner.

3.2 DLR Robots Library

The DLR Robots Modelica library was the first one to use Lua to control the movements of a simulated robot (Bellmann, Seefried, and Thiele 2020). However, in the version presented then, the library was an integral part of the robot controller C code and could not be used for other purposes. With the separation of the Lua interpreter code in a stand-alone Modelica library, the Robots library had to be adapted to utilize this new generalized approach. The basic principle for controlling the robot stays the same:

A movement command, e.g. a point-to-point (PTP) request, sets a status flag (Robot Command State) and the desired target position in the virtual robot controller, the Modelica model simulates the movement, while the Lua script waits until the execution of the command is finished (Figure 6).

The major difference now is that the data is no longer stored in a special robot controller code but in the data storage of the ModelicaLua interpreter external object. Furthermore, the commands for the robot are no longer hard-coded functions defined in the compiled C interpreter but now defined in a small Lua script using the commands from Table 1. For example, the command to move a robot on a Cartesian path from its current position to the position `(x,y,z,A,B,C)` is defined like this:

```
Robots.ptpCartesianSpace =
function(x,y,z,A,B,C)
 Modelica.setReal('brl_pos_ref',
    {x,y,z,A,B,C})
 Modelica.setInteger(
    'brl_robotCommandState',
    Robots.PTPCARTESIAN)
 Robots.waitForRobot()
end
```

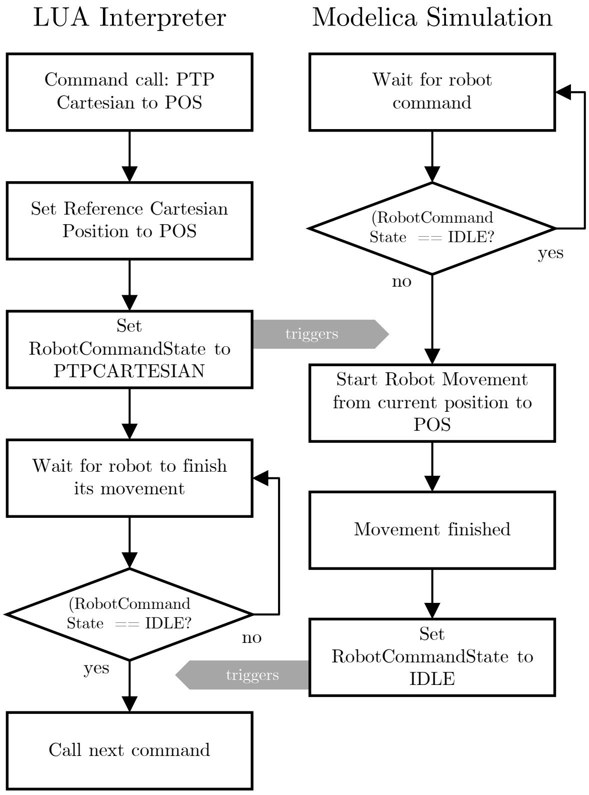

LUA Interpreter
Modelica Simulation
Command call: PTP Cartesian to POS
Set Reference Cartesian Position to POS
Set RobotCommandState to PTPCARTESIAN
Wait for robot to finish its movement
(RobotCommand State == IDLE?
no
yes
Call next command
Wait for robot command
(RobotCommand State == IDLE?
yes
no
triggers
Start Robot Movement from current position to POS
Movement finished
Set RobotCommandState to IDLE
triggers

Figure 6. Principle of how the execution of movement commands is performed, from (Bellmann, Seefried, and Thiele 2020).

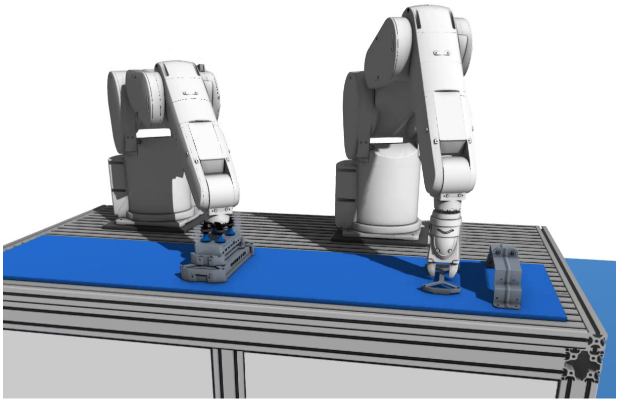


Figure 7. Two robots performing an assembly task programmed by Lua scripts, see (Reiser 2021)

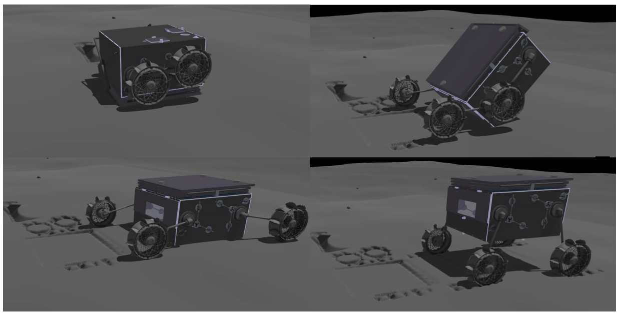


Figure 8. Visualization of the MMX up-righting sequence, shown from top left, to bottom right

This allows the user to easily integrate new commands in the robot controller script language, or modify the existing one. Figure 7, from (Reiser 2021), shows a simulation visualization of two Mitsubishi RV7-FL robots, performing an assembly task. In this use-case, two Lua interpreters run in parallel, each providing a single robot with the movement commands to perform its task. Additionally, Lua script commands are used to operate the robot tools, e.g. controlling the gripper.

3.3 MMX Rover Development

A second example for the usage of the Lua library is in the development of the MMX rover (Ulamec et al. 2019; Bertrand et al. 2019; Buse et al. 2021). This rover, jointly developed by CNES and DLR, will fly with JAXA's MMX mission to the Martian moon Phobos. There, it will be dropped onto the surface and will come to rest in a random orientation. Once it has come to rest, it has to unfold its legs in the correct order, to reorient itself on its belly and stand up. This sequence, called up-righting by the development team, must work reliably in a variety of situations. A simple example of the rover up-righting itself from its back to the belly is shown in Figure 8.

The algorithm for controlling the up-righting sequence must be developed and tested. Implementing this complex, nested state machine with the Modelica state machine would be complicated and would also require recompiling the model after each change. With the Modelica Lua library, it is possible to test different variations of the algorithm quickly. Since the script is loaded from disk, the same compiled model of the rover system can be used to launch a multitude of instances with different, random initial conditions to analyze the algorithm's robustness. Mechanical components of the rover as well as its interaction with the environment are modeled with the DLR Rover Simulation Toolkit (Hellerer, Barthelmes, and Buse 2017).

The Lua code used in this application performs multiple tasks. First, the control functions specific to the rover itself are abstracted in a separate Lua library. This enables a very direct and comprehensive approach to program the rover actions. Based on this, the high level logic of the rover is modeled as a state machine using the same approach as in the example above. The state machine controlling the rover has 36 unique states and 46 transitions in total. The `StandUp` state is used as an example here. This state becomes active once the rover successfully reoriented itself onto its belly and is now ready to stand up. This corresponds to the transition from bottom-right to bottom-left in the Figure 8. When the state becomes active, the `onStandUp` function is called once, with the interface of the rover abstracted into `Rover` a predefined angle and velocity is commanded:

```lua
onStandUp = function()
  Rover:setTargetLegVelocity(
      Rover.standupVelocity)
  Rover:setTargetLegAngles(
      Rover.legStandingAngles)
end
```

While the `StandUp` state is active, the `runStandUp` function is called periodically, here a timer and the rover interface are used to wait 15 seconds once the legs have reached the commanded target:

```
runStandUp = function()
 if not Rover:legsAtTarget() then
  timer:reset()
 elseif timer:elapsed() > 15 then
  uprighting:Standing()
 end
end
```

Once this condition is met, the next transition in the state machine is triggered by `uprighting:Standing()`.

Other than controlling the rover itself, the Lua code also reads initial configuration parameters and logs internal states to disk. With the results of both Modelica and the Lua scripts, a statistical analysis of the systems behavior is performed. For example, causalities between specific situations in the mechanical system and behavior of the control logic could be identified.

4 Conclusions

The presented Modelica Lua library allows easy and fast integration of high-level logic into Modelica. The simple interface makes integration into existing models easy. The library developed from the initial implementation in the DLR Robots library has been extended and generalized and has proven its usefulness especially in the development of the MMX up-righting algorithm. By providing access to existing third-party libraries in Lua, a language widely used in game development, a wide range of powerful tools is now available to be used in Modelica. It is planned to release this library with an open source license to make it available to the public.

References

Bellmann, Tobias, Andreas Seefried, and Bernhard Thiele (2020). "The DLR Robots library – Using replaceable packages to simulate various serial robots". In: *Proceedings of the Asian Modelica Conference 2020*. DOI: 10.3384/ecp2020174153.

Bertrand, Jean et al. (2019). "Roving on Phobos: Challenges of the MMX Rover for Space Robotics". In: *Proceedings of 15th Symposium on Advanced Space Technologies in Robotics and Automation*.

Bjorn (2021). *Lua vector math library*. https://github.com/bjornbytes/maf. [Online; accessed 3-May-2021].

Blockwitz, Torsten et al. (2012). "Functional mockup interface 2.0: The standard for tool independent exchange of simulation models". In: *Proceedings of the 9th International Modelica Conference*. DOI: 10.3384/ecp12076173.

Buse, Fabian et al. (2021). "Wheeled locomotion in milligravity: A technology experiment for the MMX Rover (accepted)". In: *72th International Astronautical Congress*. International Astronautical Federation.

Conroy, Kyle (2021). *LUA state machine library*. https://github.com/kyleconroy/lua-state-machine. [Online; accessed 3-May-2021].

Hellerer, Matthias, Stefan Barthelmes, and Fabian Buse (2017). "The DLR Rover Simulation Toolkit". In: *Proceedings of Advanced Space Technologies in Robotics and Automation 2017*. ESA's Automation and Robotics group.

Reiser, Robert (2021). "Object Manipulation and Assembly in Modelica". In: *Proceedings of the 14th International Modelica Conference 2021*.

Ulamec, Stephan et al. (2019). "A rover for the JAXA MMX Mission to Phobos". In: *70th International Astronautical Congress*. International Astronautical Federation.

Object Manipulation and Assembly in Modelica

Robert Reiser[1]

[1]Institute of System Dynamics and Control, German Aerospace Center (DLR), Germany,
{firstname.lastname}@dlr.de

Abstract

This paper introduces a new library for the manipulation and assembly of 3D objects using Modelica. The method is based on collision detection, contact dynamics, position and orientation control as well as states for object manipulation. The aim is a stable and efficient simulation of these processes close to real physics. 3D objects (both elementary shapes and CAD objects) can be added from the library browser to a model by drag-and-drop and are directly capable of being manipulated by multiple grippers and conveyors or being assembled to an assembly. This allows a high degree of flexibility and the modeling effort can be decreased significantly.

Keywords: manipulation, grasping, gripper, assembly, collision detection, contact dynamics, state machine

1 Introduction

The manipulation and assembly of objects is widely used in multiple domains (e.g. in the production industry). This leads to a high relevance for the modeling and simulation of these processes. However, current simulation tools are either specialized standalone software with restricted flexibility (Miller and Allen 2004; León et al. 2010) or embedded in plant planning software and therefore limited to this specific domain (Kühn 2006). On the other hand, the object-oriented modeling language Modelica (Modelica Association 2017) offers cross-domain flexibility.

In the work of Ferretti et al. (2006), a gripper was modeled and simulated using the Modelica Multibody Library (Otter, Elmqvist, and Mattsson 2003). The Modelica Contact Library of Oestersötebier, Wang, and Trächtler (2014) aims at the idealized simulation of contacts with non-central contact blocks. In both works, the contact pairs are pre-defined in the model. Therefore, an object cannot be grasped by multiple grippers and can only be placed on the same ground. Elmqvist et al. (2015) introduced a framework for multibody contacts in Modelica using the discrete element method where objects interact without pre-defined connections. However, due to their complexity, all models are not well suited for real time simulations.

The new solution presented in this paper aims at the stable simulation of manipulation and assembly processes in real-time close to real physics. The contact dynamics model is therefore simplified and only used for the contact between object and ground or conveyor and the initial contact between object and gripper jaws. The stable contact between object and gripper or within assemblies is achieved by a "fixed connection" (nstead of contact forces and torques (see subsection 3.3). 3D objects can be added to a model without any pre-defined connection and are directly capable of being manipulated or assembled.

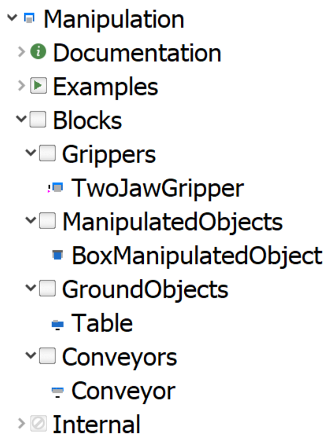
The developed library (which is currently only used internally) combines the flexibility of a non-causal, object-oriented Modelica multi-body environment with the performance of a collision detection algorithm in a linked C library. Precise simulations of the contact and frictional forces or multipoint contacts are out of the scope of this work. However, the library is modular and can be extended in multiple ways, e.g. adding precise physical models for object contact.

The following section gives an overview of the library including the blocks which can be used for modeling and information about the usage of the library. Section three deals with the models and algorithms including contact detection, contact dynamics, hold control and manipulation logic. The next section presents the concepts of object manipulation and assembly.

In addition, two applications are shown, one for the object manipulation and one for the assembly of an object. In conclusion, the advantages and disadvantages of the solution as well as future developments are discussed.

2 Overview

The first part of this section is about the available blocks and the second part about the usage of the library. Figure 1 shows the structure of the library.



Manipulation
Documentation
Examples
Blocks
Grippers
TwoJawGripper
ManipulatedObjects
BoxManipulatedObject
GroundObjects
Table
Conveyors
Conveyor
Internal

Figure 1. Overview of the library structure.

2.1 Blocks

The available blocks are shown in Figure 2 (visualization) and 3 (model) and introduced in the following subsections.

2.1.1 Grippers

The **`TwoJawGripper`** is a simple gripper with two jaws (it is possible to model grippers with more than two jaws). It can be attached to a robot. The following parameters can be set by the user. `vMax` is the maximum speed of the jaws for closing and opening. `jawPenMax` defines the maximum penetration depth for the jaws. Once the jaws have contact to the manipulated object, the velocity is reduced in relation to the penetration until the given maximum penetration is reached. The `graspDuration` is the time period after which the grasping transitions from jaw contact to stable grasping and therefore the jaws will stop the movement. There are two input states: close (jaws move inwards with the maximum velocity) and open (jaws move outwards with the same velocity). The gripper only contains a kinematic model, i.e. the gripper and its jaws are free of forces and torques.

2.1.2 Manipulated objects

The main object is the **`BoxManipulatedObject`**. It is a cuboid whose dimensions `dim` can be defined by the user. Additional parameters are the start position `rStart`, the start orientation `anglesStart`, the mass `m` and the inertia tensor as well as visualization properties. If `useJawContact` is true, the contact forces and torques caused by the jaws are used for the duration defined in `graspDuration` (only for the object and not for the gripper). If false, the object is directly attached to the gripper (close to real physics, see section 3).

The object can be used both as cuboid and as CAD object. For the latter, the CAD object is only used for visualization, i.e. the cuboid is still used for contact. Therefore, it is possible to manually set the surface for grasping (e.g. only a part of the CAD object is covered by the cuboid).

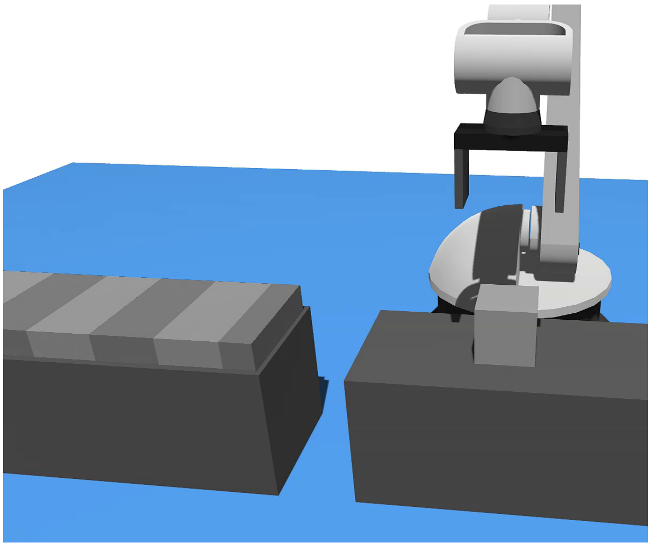


Figure 2. Visualization of Modelica blocks for object manipulation and assembly: `TwoJawGripper` (attached to robot), `Table` (bottom right), `BoxManipulatedObject` (on top of the table) and `Conveyor` (left).

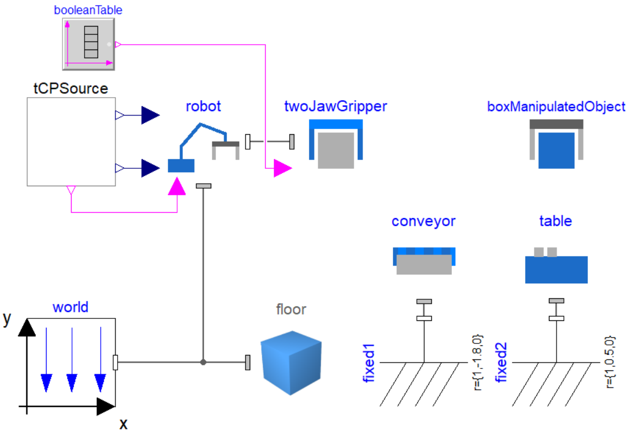


Figure 3. Modelica model with robot, `TwoJawGripper`, `Table`, `BoxManipulatedObject` and `Conveyor`.

2.1.3 Ground objects

The **`Table`** can be used as a ground object and is parameterized by the dimensions `dim` and visualization properties. The `BoxManipulatedObject` can be placed on a `Table`, which does not use forces and torques.

2.1.4 Conveyors

For the movement of objects, the **`Conveyor`** model can be used. It is also free of forces and torques and represents a moving belt which leads to a relative velocity of the `BoxManipulatedObject`. The resulting friction forces and torques move the object. The user can set the parameters `units` (number of belt elements), `velocity`, the belt dimensions, the frame dimensions and visualization properties. It can be placed horizontally or oblique to the ground. It is capable of moving objects upwards within a angle which depends on the contact parameters set for the `BoxManipulatedObject`.

2.2 Usage

Blocks can be added from the library browser to a model by drag-and-drop. There are no connections between the objects in the model because contacts are not pre-defined. All objects are directly capable for manipulation or assembly. For the `BoxManipulatedObject`, the start position and the start angles have to be defined. The initial position must be on top of a `Table` or `Conveyor`. An initialization within a closed gripper is planned, but not yet implemented. The parameter `useJawContact` is true by default.

The `TwoJawGripper` has to be attached to a robot. If the `graspDuration` (i.e. the period after the activity transitions from jaw contact forces to stable grasping) is changed, it must be changed both for the gripper and the `BoxManipulatedObject`.

3 Models and algorithms

This section shows the main models and algorithms used for the manipulation and assembly of objects. At first, the contact detection concept is introduced. In addition, the models for object forces and hold control are explained. The last subsection contains the manipulation logic.

3.1 Contact detection with libccd

The contact detection of the presented Manipulation library is done by **libccd**, a library for the calculation of collisions and penetrations between objects (Fiser 2018). It is written in C, open source under the 3-clause BSD license (Open Source Initiative 2020) and implements the following contact detection algorithms:

- Gilbert-Johnson-Keerthi (GJK)
- Expand-Polytope-Algorithm (EPA)
- Minkowski Portal Refinement (MPR)

For the purpose of this paper, the MPR algorithm is used because it is more stable.

Minkowski Portal Refinement is a collision detection algorithm developed by Gary Snethen (Snethen 2008). It is similar to the GJK algorithm.

Both are limited to convex shapes and use the Minkowski difference which can be described as *"the region swept by Object A translated to every point negated in Object B"* (Serrano 2016). It is widely used in collision detection because the location of the origin indicates if both objects collide. If the Minkowski difference includes the origin, there is a collision. (Serrano 2016)

The MPR algorithm also uses the Support Function, which is a *"function that takes a convex hull (or convex polygon) and a support vector representing a direction of search. The function returns the point on the body that is farthest in the sense of a plane sweep using the support vector"* (Newth 2013).

The MPR algorithm in 2D can be described as follows (Snethen 2008; Newth 2013) and is shown in Figure 4. In general, it consists of two phases: portal discovery and portal refinement.

The **portal discovery** starts from the Minkowski difference between two objects and the origin `O` (Figure 4a).

- The phase begins with the definition of an interior point `V0` within the Minkowski difference (e.g. the geometric center). The origin ray is then defined as the ray from `V0` to `O` (Figure 4b).
- With the origin ray, the first support point `V1` is determined based on the support function. The second support point `V2` is discovered by a ray perpendicular to the vector from `V0` to `V1` (Figure 4c, 4d).
- The support points `V1` and `V2` are now defining a portal. If the origin passes through this portal, the algorithm continues with the second phase. If not, the process above continues with finding a new support point in normal direction to the portal (Figure 4e, 4f).

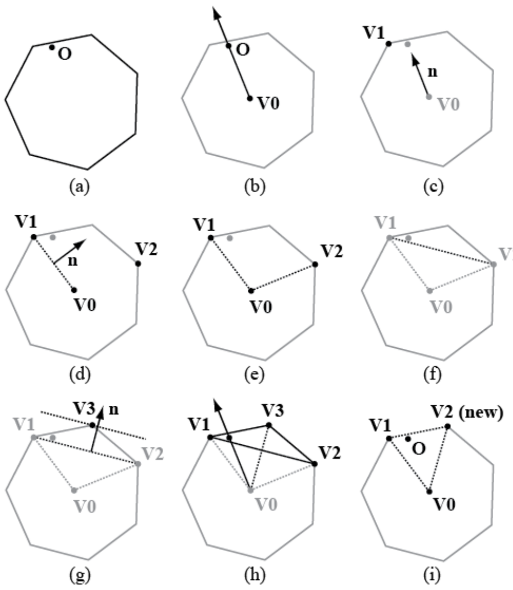


Figure 4. Phases of the MPR algorithm (Snethen 2008). The initial portal is discovered so that the ray from the interior point V0 to the origin O passes through (V1 and V2 define the initial portal). This portal is then refined until the origin is on the inside of the portal (V1 and V2 (new) define the final portal). The shape represents the Minkowski difference between two objects.

The **portal refinement** starts from an initial portal.

- It is checked, if `O` is inside the portal. If this is not the case (Figure 4f), the portal will be refined.
- A normal perpendicular to the current portal is created to find the next support point `V3` and a new portal is created (Figure 4g, 4h).
- This step is repeated until `O` is inside the portal (Figure 4i).

The MPR algorithm in 3D is similar with the portal becoming a triangle instead of a line segment. Based on the MPR algorithm, libccd is able to determine if two objects collide and if so to calculate the position, direction and maximum depth of the collision.

The interface between libccd and Modelica is based on unpublished work of Hellerer (2019). With this library it is possible to define the mentioned object types in Modelica and perform collision detection in Modelica based on libccd.

In addition, the unpublished library of Buse (2021) adds collision groups and the memory handling on C, which enables the usage of collision objects in Modelica models by drag-and-drop.

In the context of object manipulation, the contact detection is limited to a single contact, i.e. an object can collide

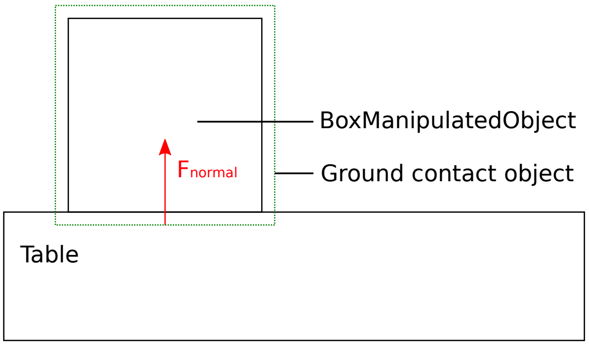


Figure 5. Contact between `BoxManipulatedObject` and `Table` based on the contact dynamics model. The `BoxManipulatedObject` contains an object for ground contact (green dotted, invisible in simulation visualization) with larger dimensions to compensate the lower stiffness necessary for fast simulations.

with multiple other objects, but only the contact with the maximum depth is transferred to Modelica.

3.2 Contact dynamics model

This model is used for forces and torques between objects, both for ground contact (object is placed on a table or conveyor) and jaw contact (interaction between jaws and the grasped object).

The libccd uses support functions which are defined by the geometry of the shapes during the initialization of the simulation. During simulation, the current position, orientation and velocity of all collision objects are constantly transferred to the libccd. In return, information about the collision are transferred to Modelica. These are the position, direction and depth of a collision.

The contact dynamics model is simplified so that a fast and stable simulation close to real physics is possible. The normal force is based on a spring-damper-model:

$$F_{normal} = k \cdot s_{penetration} + d \cdot v_{penetration} \tag{1}$$

where $s_{penetration}$ is the penetration depth and $v_{penetration}$ the velocity of the penetration. The spring constant k and the damping constant d are set by the user. For the calculation of the friction force, the simplified Coulomb friction model from (Andersson, Söderberg, and Björklund 2007) is used:

$$F_{friction} = \mu \cdot F_{normal} \cdot \mathrm{sign}(v_{tangential}) \tag{2}$$

which is simplified to

$$F_{friction} = \mu \cdot F_{normal} \cdot \tanh(k_{tanh} \cdot v_{tangential}) \tag{3}$$

where μ is the coefficient of friction and $v_{tangential}$ the tangential velocity. For simulation performance, sign() is replaced by tanh(), scaled by the coefficient k_{tanh}.

Since hard contacts decrease the performance of the simulation significantly, it is possible to use soft contact parameters, i.e. lower stiffness (e.g. for ground contact). To prevent objects from sinking in the ground when soft contacts are used, the dimensions for the contact object can be extended which compensates for the lower stiffness and leads to visualizations close to reality. The dimension for the ground contact object dim_{ground} is calculated by:

$$dim_{ground} = dim + \frac{m \cdot g}{k} \cdot \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix} \tag{4}$$

where dim is the dimension, m the mass and k the the spring constant of the `BoxManipulatedObject`.

The `BoxManipulatedObject` contains three contact dynamics models for ground and both jaw contacts, each with its own collision object. Therefore, the contact parameters can be set specifically for each contact. Figure 5 shows the `BoxManipulatedObject` and its collision object for ground contact. The larger dimension of the ground contact object compensates the lower stiffness of the contact. The jaw contact objects are similar to the ground contact object (see Figure 6). Each contact object generates a force and torque which are summarized in the `BoxManipulatedObject`.

3.3 Hold control

The hold control generates a "fixed connection" between two objects and is used for stable grasping and stable assembly. The concept consists of two objects:

- A low-level object (LLO) which is held by the top-level object. It contains a controller for position and orientation control.
- The top-level object (TLO) which is only necessary for collision detection.

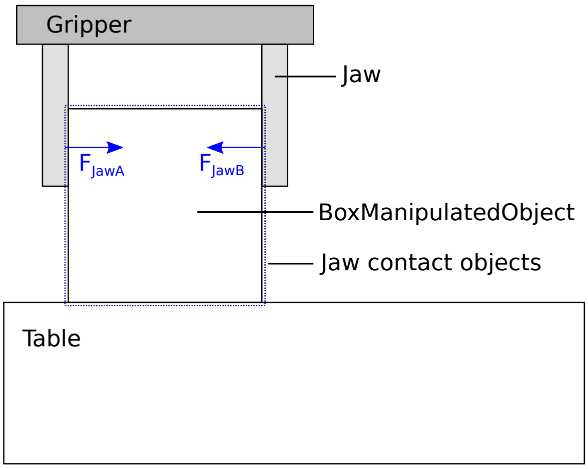


Figure 6. Contact between `BoxManipulatedObject` and `TwoJawGripper` based on the contact dynamics model. The `BoxManipulatedObject` contains one object for each jaw contact (blue dotted, invisible in simulation visualization).

DOI 10.3384/ecp21181433

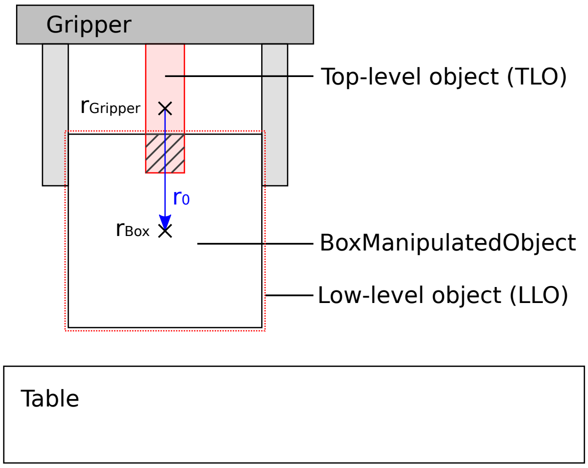


Figure 7. Hold control used for the gripper. The LLO (invisible in simulation visualization) gets the position and orientation from the TLO (invisible in simulation visualization). Based on the initial difference (r_0), the target position and orientation is calculated and a PI-controller is used to keep the object there.

The procedure can be described as follows:

At first, the collision detection is disabled for the TLO. Then it is moved into the LLO to get a penetration. From the moment the LLO object should be held in place by the TLO (i.e. the gripper closes), the ability of the TLO to collide is activated. The LLO recognizes a collision and stores the initial difference in position and orientation between both objects:

$$r_0 = \mathrm{T}_{\text{Gripper}} \cdot (r_{Box} - r_{Gripper}) \tag{5}$$
$$\mathrm{T}_0 = \mathrm{T}_{\text{Box}} \cdot \mathrm{T}_{\text{Gripper}}{}^T \tag{6}$$

where r_{Box} is the position and T_{Box} the rotation matrix of the box. From the moment the gripper begins to close, the LLO recieves continuously the position ($r_{Gripper}$) and orientation ($\mathrm{T}_{\text{Gripper}}$) of the TLO. Based on this information and the stored initial difference, it is possible to calculate the target position and orientation for the object:

$$r_{Target} = r_{Gripper} + \mathrm{T}_{\text{Gripper}}{}^T \cdot r_0 \tag{7}$$
$$\mathrm{T}_{\text{Target}} = \mathrm{T}_0 \cdot \mathrm{T}_{\text{Gripper}} \tag{8}$$

A PI-controller and a gravity compensation for the weight are finally used to keep the LLO in the target position and orientation.

The forces and torques are only present in the low-level object: in the grasped object (not in the gripper) and in the sub-assembly (not in the assembly).

Figure 7 shows the application of the hold control model for a gripper. The TwoJawGripper contains a TLO. It is invisible in the visualization and penetrates the BoxManipulatedObject, which contains the LLO.

3.4 Manipulation logic

The manipulation logic is used to switch between the states of a BoxManipulatedObject. There are the following states (see Figure 8):

1. GroundContact (initial state): The object can be placed on a ground object (e.g. Table) or a Conveyor.

2. JawContact (grasping part with jaw interaction until hold control is used): Both ground contact forces and jaw contact forces are used. This state is only used if useJawContact is true.

3. Grasping (based on hold control): The object is attached to the TwoJawGripper based on hold control for position and orientation. The transition from state 2 to state 3 is done if the graspingTime is over or if the grasping is stable (minimum relative rotation and movement between jaw and object).

4. Assembly: Is used if the object is placed in an assembly and grasping is over. Grasping is always higher prioritized than Assembly.

5. Delay (between Assembly and Grasping): This is necessary for the following condition:

   - The object is part of an assembly.
   - Assembly and object are grasped by a gripper.
   - useJawContact is enabled.

   In this case, the object would normally be instantly controlled by the gripper. This delay holds the object in Assembly state until JawContact for the assembly is done. This state is necessary, if e.g. the basis of an assembly is slightly rotated by the jaws: all parts of the assembly have to be rotated as well.

The transitions between the states (shown in Figure 8) are defined as follows:

```
a := graspingActive and useJawContact and
     not stableGrasping and not
     assemblyActive
b, c, f, g := graspingActive and (not
     useJawContact or useJawContact and ((
     stableGrasping or (time >=
     graspInitTime + graspDuration)))
```

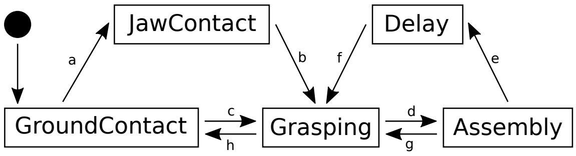


Figure 8. Manipulation states and their transitions.

```
d := not graspingActive and assemblyActive
e := graspingActive and useJawContact and
    not stableGrasping and assemblyActive
h := not graspingActive and not
    assemblyActive
```

4 Object manipulation and assembly

In this section, the concepts of object manipulation and assembly are shown. This includes the placement of objects on ground and on conveyors, grasping objects with grippers and the assembly of objects.

4.1 Object placement on ground

A `BoxManipulatedObject` contains a ground contact object with the underlying contact dynamics model. Based on this, the object can be placed on a `Table` as mentioned in subsection 3.2 and shown in Figure 5.

4.2 Object transport with conveyors

The same model is used if the object is placed on a `Conveyor`. The only difference is the velocity of the `Conveyor` which leads to a movement of the `BoxManipulatedObject` (see subsubsection 2.1.4).

`Conveyors` can be attached together. The subsequent conveyor has to be slightly lower relative to the previous one to avoid the `BoxManipulatedObject` from being stuck between both `Conveyors`.

4.3 Object grasping with grippers

An object can be grasped by all `TwoJawGrippers`. The `BoxManipulatedObject` contains two jaw contact objects to apply forces and torques from both gripper jaws (see subsection 3.2).

In addition, hold control is used for a stable connection between object and gripper as explained in subsection 3.3.

A transfer of a `BoxManipulatedObject` from one `TwoJawGripper` to another is possible to cover a wide range of applications.

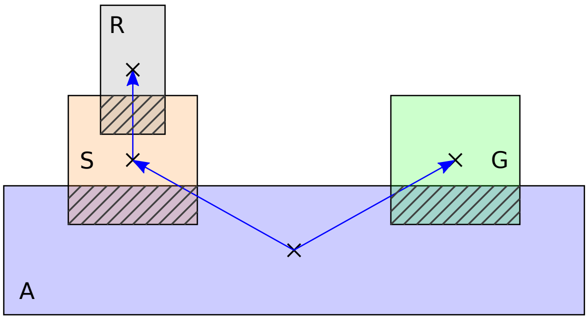


Figure 9. Assembly with sub-assembly. R is assembled to S and both S and G are assembled to A. The vectors are showing the position differences between the object origins.

4.4 Object assembly

The assembly of objects is based on the same principle as grasping, namely the hold control concept introduced in subsection 3.3. The `BoxManipulatedObject` contains a TLO and a LLO for assembly.

If two objects are assembled, one object must be defined as base and the other one as module. The user has to enable the TLO for the base and the LLO for the module. Then it is possible to "attach" the module to the base.

The same applies if two modules are assembled to a base. Both modules need an activated LLO and the base represents the TLO for the hold control model.

For the representation of sub-assembly processes, the definition of the TLO and LLO is more complicated. Therefore, collision groups are used. For each collision object, `assemblyCollisionGroups` and `assemblyIncludedGroups` can be defined.

The concept is demonstrated with an example containing four objects: gray rod (R), orange sub-assembly box (S), green box (G) and main assembly (A) (see Figure 9).

The following collision groups are defined:

```
R.assemblyCollisionGroups = {""}
R.assemblyIncludedGroups = {"SubAssembly"}
S.assemblyCollisionGroups = {"SubAssembly"}
S.assemblyIncludedGroups = {"Assembly"}
G.assemblyCollisionGroups = {""}
G.assemblyIncludedGroups = {"Assembly"}
A.assemblyCollisionGroups = {"Assembly"}
A.assemblyIncludedGroups = {""}
```

This means:

- R is a LLO in relation to S
- S is a TLO in relation to R and a LLO regarding A
- G is a LLO in relation to A
- A is a TLO in relation to S and G

Figure 9 shows the assembly for the example. The vectors are showing the initial position differences used for the hold control model. To manipulate an entire assembly, the `TwoJawGripper` can only grasp the base object and not the sub-assemblies. If a `TwoJawGripper` grasps a sub-assembly, it can be disassembled.

5 Applications

In this section one example for object manipulation and one for assembly are shown. The simulations are visualized with the DLR Visualization 2 Library (Kümper, Hellerer, and Bellmann 2021).

5.1 Object manipulation

In this example, objects are manipulated with grippers (attached to robots) and `Conveyors`. The model is shown in Figure 10 and the visualization illustrated in Figure 11. There are the following objects:

- Orange box on front table (O)

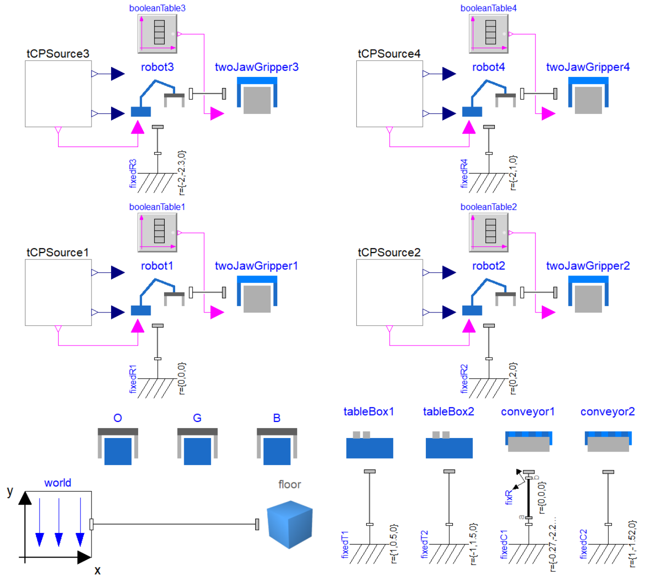


Figure 10. Modelica model for the manipulation of three objects (O, G and B) with robots, grippers and conveyors.

- Green box on front table (G)
- Blue box on back table (B)

At first, the robot in the front grasps object G and places it on the right of the same Table. The initial orientation of G is rotated 8 degrees to the lateral surface of the Table. Since the gripper is held in an orthogonal orientation above the object, the jaws force the object into the orientation of the gripper. Therefore, G will be placed in a perfect orientation on the Table. Meanwhile, the robot in the back moves B to the position between the two robots in the back. The object is then handed over to another robot (Figure 11 (middle)), while G is moved to the back Table and O to the right of the Table. Object O is rotated in the beginning as well. Finally, B is placed on the Conveyor and moves back to the front, onto the other Conveyor and then to the Table. The end of the sequence is shown in Figure 11 (bottom).

The model was tested in Dymola 2020 (64-bit) on Windows 10 on a Intel® Xeon® W-2135 workstation. A Rkfix2 solver with the fixed step of 0.001 was used. The "CPU-time for integration" for the 40 s simulation was 19.1 s. Therefore the model is real-time capable.

5.2 Assembly of an object

This example is the same as mentioned in subsection 4.4. It shows the assembly of a module cosisting of multiple BoxManipulatedObjects with TwoJawGrippers and robots including a subassembly. The model is shown in Figure 12. The initial setting is visualized in Figure 13 (top). There are the four objects R, S, G and A.

The first step is to assemble the rod into the subassembly box. This is done by the left robot. Then both S and G are assembled to the main assembly by both robots. Finally, the right robot is able to grasp the entire assembly as shown in Figure 13 (bottom) and to move it to the second Table in the back.

In this example, the hold control model is used for grasping and for assembly. R is "attached" to S in the same way as a box is "attached" to a gripper. S and G are connected to A similarly.

The same settings as in subsection 5.1 are used to simulate the model. The 40 s simulation took 14.5 s for integration, i.e. the model is real-time capable as well.

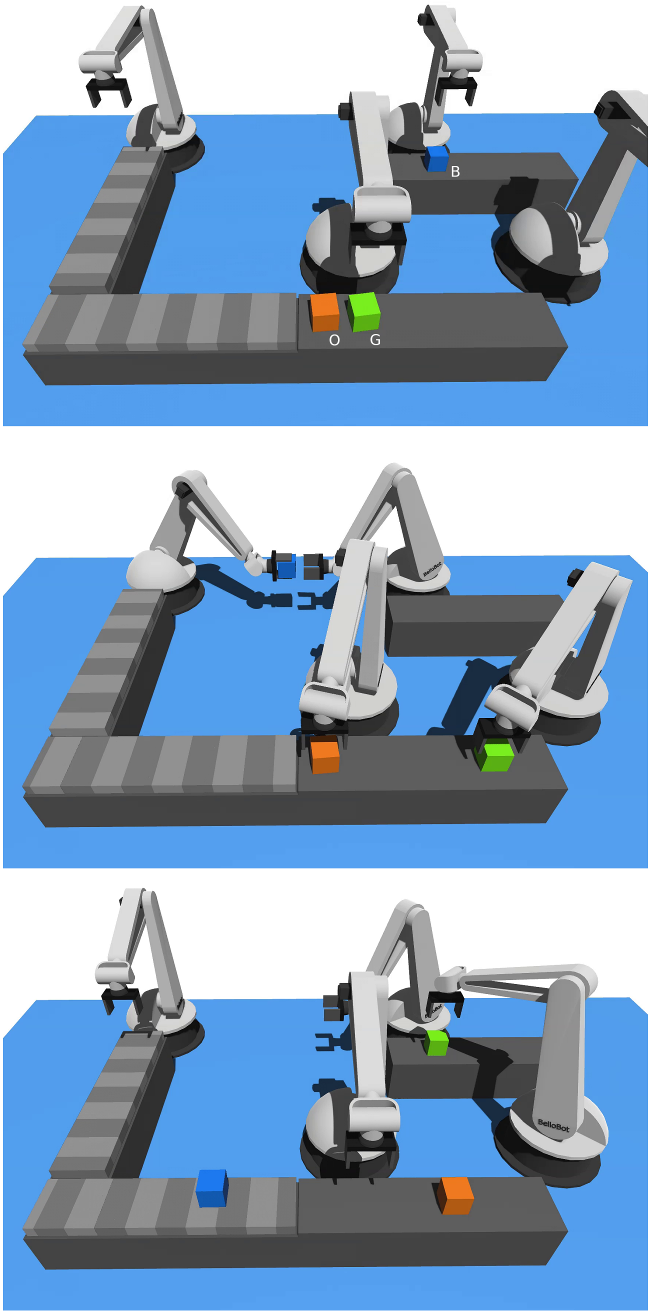


Figure 11. Manipulation of multiple objects with Grippers and Conveyors with start (top), object handed over from one robot to another (middle) and end of sequence (bottom).

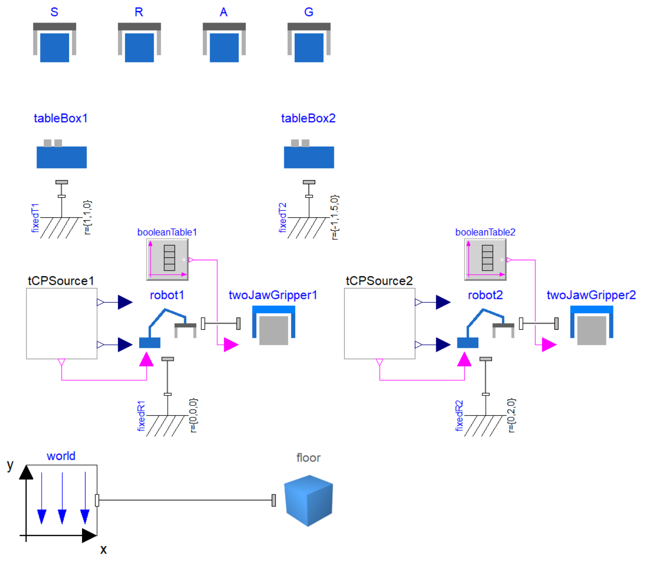


Figure 12. Model for the assembly of the objects S, R, A and G.

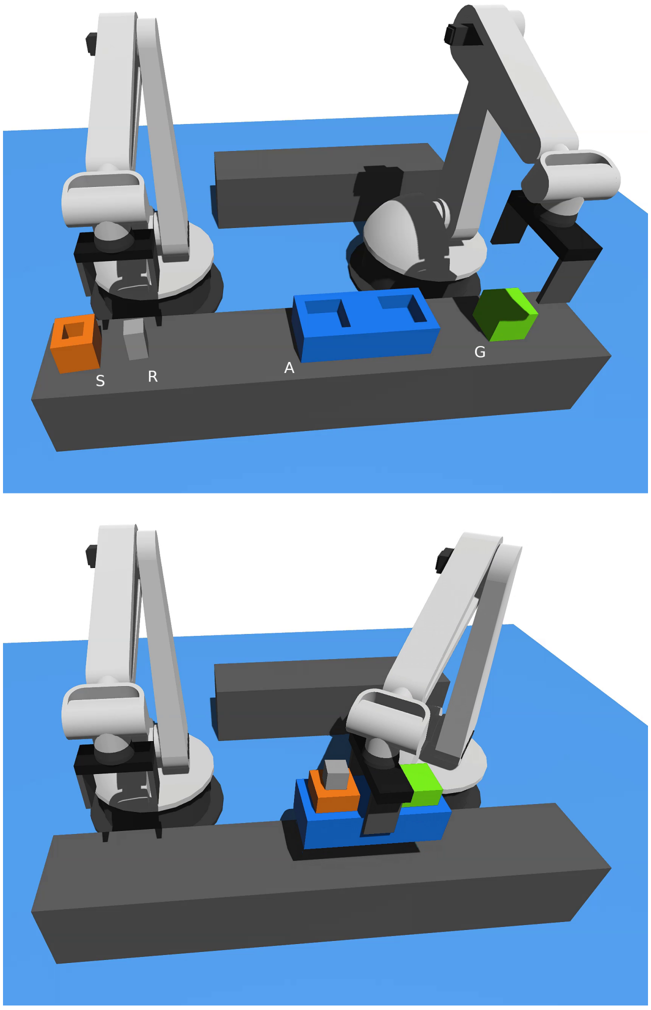


Figure 13. Assembly of an object including a sub-assembly. The grey rod is assembled to the orange box and both the sub-assembly and green box are assembled to the blue base box.

6 Discussion

The successful application of the library for object manipulation and assembly was demonstrated in the previous section. Both mentioned models (Figure 10, 12) are real-time capable. The simulation speed depends on the number of manipulated objects in the model. Real-time simulations are currently possible for models with up to 20 objects (depending on the number of robots, grippers, conveyors and tables in the model). However, the library still has restrictions and limitations:

- The contact detection algorithm is limited to one contact, therefore an object can be placed on a table but it is not possible to place another object on top of the former object.
- If an assembly is grasped by a gripper, both the gripper and the base-object of the assembly must have contact (i.e. it is not possible to grasp an assembly by grasping a sub-assembly of the assembly).
- There are no forces and torques on top level. Therefore, it is not possible to use the library for robot dynamic simulations.

Possible future developments are:

- Forces and torques for the top-level object (e.g. gripper) in the hold control model.
- Maximum hold force in the gripper. If the force exceeds the maximum, the connection will be disabled.

7 Conclusion

The combination of the non-causal, object-oriented Modelica multi-body environment and the performant MPR collision detection algorithm in C are the basis for a new solution to the modeling and simulation of manipulation and assembly processes. The real-time capability and stability were demonstrated in two use cases.

Blocks for manipulated objects, grippers, conveyors and tables can be added from the library browser to a model by drag-and-drop. This allows a low modeling effort with a high flexibility at the same time. The blocks can be combined with all libraries in the Modelica environment. Therefore, this library lays the foundation for plant simulation on multiple levels of detail.

Acknowledgements

I would like to thank Tobias Bellmann for fruitful discussions regarding the library, Fabian Buse for valuable discussions about and providing models for contact detection and contact dynamics, Matthias Hellerer and Bernhard Thiele for help with Modelica, Patrick Weber for providing the basis for the conveyor and Sebastian Kümper for help with the visualization (all DLR).

This work benefited from the MFlex 2025 project founded by the German Federal Ministry for Economic Affairs and Energy (BMWi).

References

Andersson, Sören, Anders Söderberg, and Stefan Björklund (2007). “Friction models for sliding dry, boundary and mixed lubricated contacts”. In: *Tribology International* 40.4. NORDTRIB 2004, pp. 580–587. DOI: 10.1016/j.triboint.2005.11.014.

Buse, Fabian (2021). *ContactDynamics. Modelica Library. Unpublished work.*

Elmqvist, Hilding et al. (2015). “Generic Modelica Framework for MultiBody Contacts and Discrete Element Method”. In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. The 11th International Modelica Conference. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, pp. 427–440. DOI: 10.3384/ecp15118427.

Ferretti, Gianni et al. (2006). “Modelling and simulation of a gripper with Dymola”. In: *Mathematical and Computer Modelling of Dynamical Systems* 12.1, pp. 89–102. DOI: 10.1080/13873950500071405.

Fiser, Daniel (2018). *libccd. Library for collision detection between two convex shapes*. URL: https://github.com/danfis/libccd (visited on 2020-04-28).

Hellerer, Matthias (2019). *CollisionDetection. Modelica Library. Unpublished work.*

Kühn, Wolfgang (2006). *Digitale Fabrik. Fabriksimulation für Produktionsplaner*. 1. Aufl. München and Wien: Carl Hanser Fachbuchverlag. 495 pp. ISBN: 978-3-446-40619-3.

Kümper, Sebastian, Matthias Hellerer, and Tobias Bellmann (2021). “DLR Visualization 2 Library - Real-Time Graphical Environments for Virtual Commissioning”. In: *14th International Modelica Conference 2021*. Ed. by Martin Sjölund et al.

León, Beatriz et al. (2010). “OpenGRASP: A Toolkit for Robot Grasping Simulation”. In: *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, pp. 109–120. DOI: 10.1007/978-3-642-17319-6_13.

Miller, Andrew T. and Peter K. Allen (2004). “Graspit! A Versatile Simulator for Robotic Grasping”. In: *IEEE Robotics & Automation Magazine* 11.4, pp. 110–122. DOI: 10.1109/MRA.2004.1371616.

Modelica Association (2017). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.4. Tech. Rep.* Linköping: Modelica Association. URL: https://modelica.org/documents/ModelicaSpec34.pdf.

Newth, Joshua (2013). “Minkowski Portal Refinement and Speculative Contacts in Box2D”. Master’s Thesis. San Jose State University. URL: http://www.cs.sjsu.edu/faculty/pollett/masters/Semesters/Spring12/josh/joshua_newth.pdf.

Oestersötebier, Felix, Peng Wang, and Ansgar Trächtler (2014). “A Modelica Contact Library for Idealized Simulation of Independently Defined Contact Surfaces”. In: *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*. the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, pp. 929–937. DOI: 10.3384/ecp14096929.

Open Source Initiative (2020). *The 3-Clause BSD License*. URL: https://opensource.org/licenses/BSD-3-Clause.

Otter, Martin, Hilding Elmqvist, and Sven Erik Mattsson (2003). “The New Modelica MultiBody Library”. In: *Proceedings of the 3th International Modelica Conference* (Linköping, Sweden). Ed. by Peter Fritzson, pp. 311–330. URL: https://modelica.org/events/Conference2003/papers/h37_Otter_multibody.pdf.

Serrano, Harold (2016). *Visualizing the GJK Collision detection algorithm*. URL: https://www.haroldserrano.com/blog/visualizing-the-gjk-collision-algorithm.

Snethen, Gary (2008). *Minkowski Portal Refinement in 2D. XenoCollide*. URL: http://xenocollide.snethen.com/mpr2d.html.

A Portable and Secure Package Format for Executable Simulation Modules based on WebAssembly

Moritz Allmaras*[1] Andrés Botero Halblaub[2] Harald Held[2] Tim Schenk[2]

[1]Siemens Energy Global GmbH & Co. KG, Germany, moritz.allmaras@siemens-energy.com
[2]Siemens AG, Germany, {andres.botero,harald.held,tim.schenk}@siemens.com

Abstract

We propose a new format (Digital Twin Assembly – *dtasm*) for self-contained executable co-simulation modules that is portable and sandboxed, yet offers performance close to native machine code and is sufficiently lightweight for running on embedded devices. *Dtasm* is based on WebAssembly, a standardized bytecode format for a stack-based virtual machine originally developed for high-performance computations in web browsers. A language-independent binary interface for such modules is described that is functionally comparable to FMI for co-simulation but not tied to a particular programming language. We discuss the benefits and drawbacks of this approach and how it can address some specific issues for executable simulation modules running in parallel with the operation of real systems.

Keywords: Simulation Modularization, Portability, Sandboxing, WebAssembly

1 Introduction

Recent industry trends towards digitization of production facilities, plants, infrastructure and transportation have amplified the need for digital twins not only during the design and engineering of such systems, but also for supporting their commissioning, automation and control during operation. Use cases for such digital twins performing online simulation range from virtual sensing, model predictive control and anomaly detection to optimal operation scheduling (Boschert, Heinrich, and Rosen 2018; Tao and Zhang 2017; Rasheed, San, and Kvamsdal 2020).

In contrast to the offline use of modeling and simulation during design and engineering of systems, the execution of numerical simulations in parallel to the operation of a real-world system presents some challenges that typically do not arise in offline scenarios:

- Computation needs to be sufficiently fast to keep up with the progress of the real-world system, so some kind of (soft or hard) real-time constraint needs to be fulfilled.

- The simulation needs to run robustly and reliably without human intervention for extended periods of time.

- Failure modes need to be predictable and their effects deterministic. Online simulations often perform safety critical tasks in control and automation systems for which rigorous regulations regarding testing and certification procedures are applicable.

- The hardware on which online simulations are executed is heterogeneous. Computing devices used for carrying out online numerical simulations often need to operate close to the "shop floor" of the actual physical systems to keep signal latency low. Hence, the hardware in use varies between different plants. The type of devices in use ranges from specialized microcontrollers (MCUs) to programmable logic controllers (PLCs) to industrial PCs depending on the specific application and scenario.

While the significance of each of these requirements varies from application to application, they are major contributors to the fact that online simulations in industrial applications are often customized solutions and cannot easily be re-used. Also, many established system simulation tools have their own ways of exporting online-capable simulations, often through code generation or compilation of binaries with a proprietary API (see, e.g., Schijndel (2014)), which further limits the reusability and composability of the resulting executable simulation modules.

On the other hand, as most of the industrial systems of interest are composed of smaller subsystems and components, the digital twin of such a system could also be modeled as a composition of smaller, independent subsystem or component twins. Just like for a physical asset such as a pump, gear box or conveyor belt, the same brand and model is deployed in many different real-world systems, the same should be possible for their digital twin counterparts: The digital twin of a component should be independent of the authoring tool used for its creation and be re-usable across as many different contexts and environments as possible.

In the future, components may be equipped with their own digital twins from the factory, and modularity is a major requirement for being able to integrate such supplier-provided twins into a complex system simulation. An implication of such a scenario is that the authors of digital twins are different from the plant operators running the

*Corresponding author

DOI 10.3384/ecp21181443

twins on their infrastructure. Hence, an elevated level of trust between producers and consumers of online digital twins is required, and security boundaries need to be defined that encapsulate supplier-provided twins in a safe, yet performant sandbox during their online execution.

In this text, we present a new format for self-contained executable digital twins that is both portable and sandboxed, yet allows close-to-native compute performance and is sufficiently lightweight to be used on embedded systems.

1.1 State of the Art

The modularization of industrial simulations has started to gain traction with the more widespread adoption of the *Functional Mock-Up Interface* (*FMI*) standard (Blochwitz, Otter, Arnold, et al. 2011; Blochwitz, Otter, Akesson, et al. 2012), which specifies a tool-independent interface and packaging format (*Functional Mock-Up Unit (FMU)*) for simulation modules. The *FMU* format allows system simulation tools to export self-contained simulation modules in such a way that they can be reused in different environments and by different tools than they have been authored in. FMI distinguishes *Model Exchange (ME)* and *Co-Simulation (CS)* modules, where Co-Simulation modules include a numerical solver and hence are most suitable for packaging executable simulations. FMUs may contain code as binaries ("binary FMU") or C sources ("source FMU") or both. The code contained in an FMU exposes a set of C functions that are specified by the FMI standard.

However, in the online simulation scenarios outlined above, FMI also presents some challenges regarding portability and the enforcement of security boundaries between the co-simulation master and the FMU instances:

- Binary FMUs only support the target platforms they have been explicitly compiled for, i.e. the relevant target platforms have to be known at compile time.
- Native binaries are difficult to sandbox from their executing environment. If loaded into a native process, an FMU assembly can directly interact with the OS kernel through system calls, and hence affect overall system integrity. Hence, the use of native binaries in-process requires a high level of trust in the authoring party.
- From the point of view of the embedding application, it is hard to determine upfront if a binary FMU is actually self-contained or requires additional dependencies to be dynamically linked at runtime (such as specific version of C or C++ runtime libraries). The availability of the correct version of such runtime dependencies has to be ensured though, and they are not specified in the model description.
- The runtime interface of FMI is specified in terms of C function calls, hence implementation of the interface in programming languages other than C and C++ need to rely on the *foreign function interface* (*FFI*) mechanisms of the respective programming language. While this is common practice in most programming languages, the implementation of the FMI runtime interface is often less ergonomic and safe than in C and C++.
- Source FMUs expose their internal implementation and thus are not viable in many industrial contexts where intellectual property (IP) protection is paramount.
- Source FMUs need an extra build step before they can be executed, and the FMI standard does not specify the details of this build step. Consequently, the build step is often proprietary to the generating tool and thus difficult to automate across FMUs created by different tools.

These limitations can impact the ability to exchange and re-use FMUs across different applications and hardware environments. In particular, additional measures are necessary to enforce security boundaries between the host environment and the code supplied by an FMU. An example for such measures is execution in separate processes connected through an interprocess communication (IPC) mechanism (see e.g. Hatledal et al. (2019)). However, the additional operational complexity of such multi-process setups is considerable, and on many embedded targets the necessary infrastructure and resources may not be available.

1.2 Digital Twin Assembly

Digital Twin Assembly is based on WebAssembly (Haas et al. 2017), a W3C-standardized bytecode format for a stack-based virtual machine, that has originally been developed to enable high-performance computations inside web browsers. The core WebAssembly specification (Rossberg 2019) is slim and low-level and is meant as a compilation target for compilers of high-level programming languages. Since it is independent of any other web technology, WebAssembly has recently seen increasing adoption in applications outside web browsers, such as server-side execution of user-supplied code (Hall and Ramachandran 2019), smart contract applications (Zheng et al. 2021) and Internet of Things (Jacobsson and Willén 2018). We define a simple and portable interface to such WebAssembly modules that functionally resembles FMI for co-simulation, but is not tied to specific platforms or language ecosystems.

1.3 Outline

Section 2 discusses the available options for the packaging of executable simulation modules. In section 3, WebAssembly as the target format of *dtasm* is introduced, as well as the application binary interface (ABI) that has been developed to interact with simulations packaged as WebAssembly modules, and the strengths and weaknesses of the proposed format are discussed. Section 4 deals with prototypical implementations of *dtasm* runtimes and modules that were created in the course of this work. The performance of *dtasm* is compared with that of native binaries for some exemplary cases. In section 5, we summarize our findings and give an outlook on some further topics regarding online simulation that *dtasm* could potentially help to address.

2 Packaging of Executable Simulation Modules

In this text, the definition we will use for executable simulation modules is very similar to co-simulation modules in the sense of FMI:

- Modules carry a machine-readable description of the model containing information such as model metadata, input, state and output variables, default values, validity ranges for experimental conditions and capabilities of the module's implementation.
- Modules allow the creation of independent instances of the executable simulation.
- Values for constant parameters can be supplied and initial conditions for state variables can be set.
- Progressing the module instance's state from t_i to t_{i+1} consists of the steps:
  1. Values for the input variables at time t_i are supplied to the module instance,
  2. a time step from t_i to t_{i+1} is calculated,
  3. values for output variables and states at time t_{i+1} are returned.
- The internal state of a module instance can be reset to the time step immediately preceding the current time step.
- Instances can be terminated and disposed of at any time.

The normal sequence of invocation for online simulations (without considering potential reset of timesteps) is depicted in 1. Just like for a co-simulation FMU, no restrictions on the internal implementation of the simulator are imposed. It could, e.g., implement a numerical solver for a differential algebraic equation (DAE), a forward evaluation of a trained machine learning model or even some simple table lookup mechanism.

2.1 Packaging Options

For packaging such executable simulation modules, there are two common variants:

1. Packaging native machine code targeting certain platforms.
2. Packaging the simulation's source code in some given programming language.

Binary packaging allows only the explicitly supported platforms to execute the simulation modules. On other platforms, virtualization mechanisms could be utilized, but in practice such virtualization is complex and expensive in terms of the needed compute and memory resources and hence often not a feasible option at least on embedded devices.

For source packaging, the sources need to be compiled to native machine code by the embedder of the module prior to execution on the target hardware. This allows the source code to be compiled by specialized compilers for the target hardware. In this case, the packaging format needs to specify the exact supported feature set of the programming language, as well as all necessary operations for compiling the source code to native machine code. This option places the burden of compilation on the embedder of the module, which in many cases necessitates manual intervention and prevents the automated deployment of such packages. Providing simulation's source code also exposes the intellectual property of the implementation, which is frequently a major obstacle for the adoption of such package formats in industrial contexts.

Another option somewhat in between 1 and 2 is the packaging of intermediate bytecode targeting a virtual instruction set architecture (ISA). For execution, the bytecode is then either interpreted by an application-level virtual machine or compiled to native machine code prior to execution. Well-known examples of such bytecode formats include Java bytecode, Common Intermediate Language (CIL) (as used by the Common Language Runtime of the .NET platform) and Python bytecode (used by CPython). Traditionally, bytecode formats have not received much attention as a target format for numerical computations since they are considered slow in comparison to native machine code due to the overhead incurred by interpretation or compilation. However, we believe that bytecode has some considerable advantages especially when used in online scenarios. Bytecode formats are very portable since they do not depend on a certain hardware instruction set, and they allow efficient sandboxing of executable code by limiting access to resources and intercepting system calls. However, many of the existing bytecode formats are rather complex and have explicit support for some of the high-level constructs of the corresponding ecosystem (like garbage collection). Hence, many of the existing bytecode formats are a poor fit as compile targets for system-level programming languages (like C, C++ or Fortran) that are commonly used in numerical simulation.

3 Digital Twin Assembly Format

3.1 WebAssembly Bytecode

Starting in 2015, a bytecode format for a stack-based virtual machine called WebAssembly (Wasm) has been developed by a working group of the World Wide Web Consortium (W3C). Since then, WebAssembly has reached stable version 1.0 and gained the status of a W3C-recommended standard (Rossberg 2019). Its original goal is the high-performance execution of computational logic

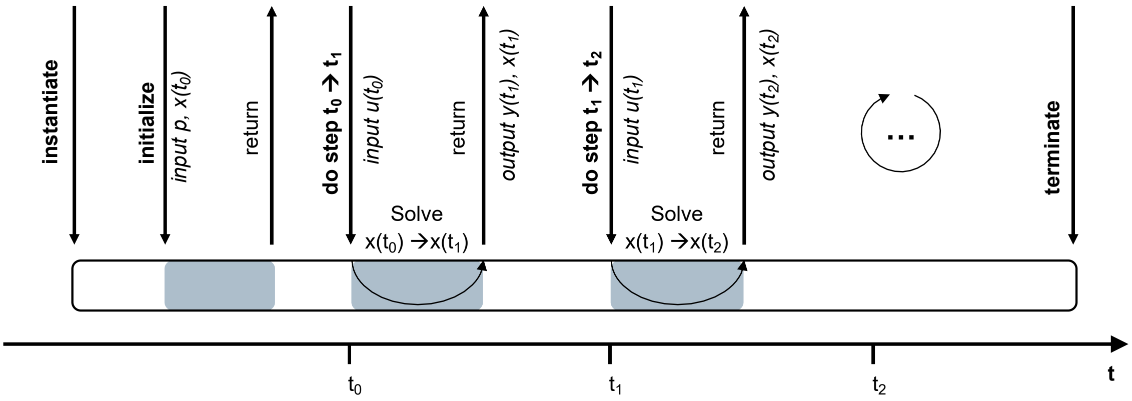


Figure 1. Lifecycle and call structure for executable simulation modules.

in web browsers. The WebAssembly specification defines a narrow low-level instruction set, with the intention that support for emitting Wasm bytecode can be easily added to existing compiler toolchains. Most notably, the LLVM compiler infrastructure (Lattner and Adve 2004) was among the first to include a backend for generating Wasm output, such that any programming language for which an LLVM frontend exists can be compiled to Wasm bytecode. In addition, WebAssembly is strongly and statically typed, and has a deterministic stack behavior that can be statically analyzed, allowing interpreters and compilers to aggressively optimize execution of the code. Consequently, WebAssembly modules can be executed with close to native performance in many scenarios (Jangda et al. 2019). The narrow instruction set also allows the implementation of lightweight interpreters for execution on small, resource-constrained devices (Peach et al. 2020). Sandboxing of the bytecode execution from the host environment has been an explicit design goal of the WebAssembly specification, since it is a paramount requirement for use in web browsers, where individual browser windows need to be kept isolated from each other and the host environment. Wasm modules cannot directly access host memory or invoke system calls. Instead, memory is provided as a contiguous linear block, and access to this block is bounds-checked by the WebAssembly runtime. System calls or calls to external libraries need to be explicitly enabled by the runtime (opt-in model) in order to be callable from inside the sandbox. WebAssembly modules are statically linked and do not (yet) support dynamic linking, so other than function imports and exports, they are self-contained. In light of the requirements for online digital twins discussed in section 1, WebAssembly offers some unique advantages over binary and source packaging:

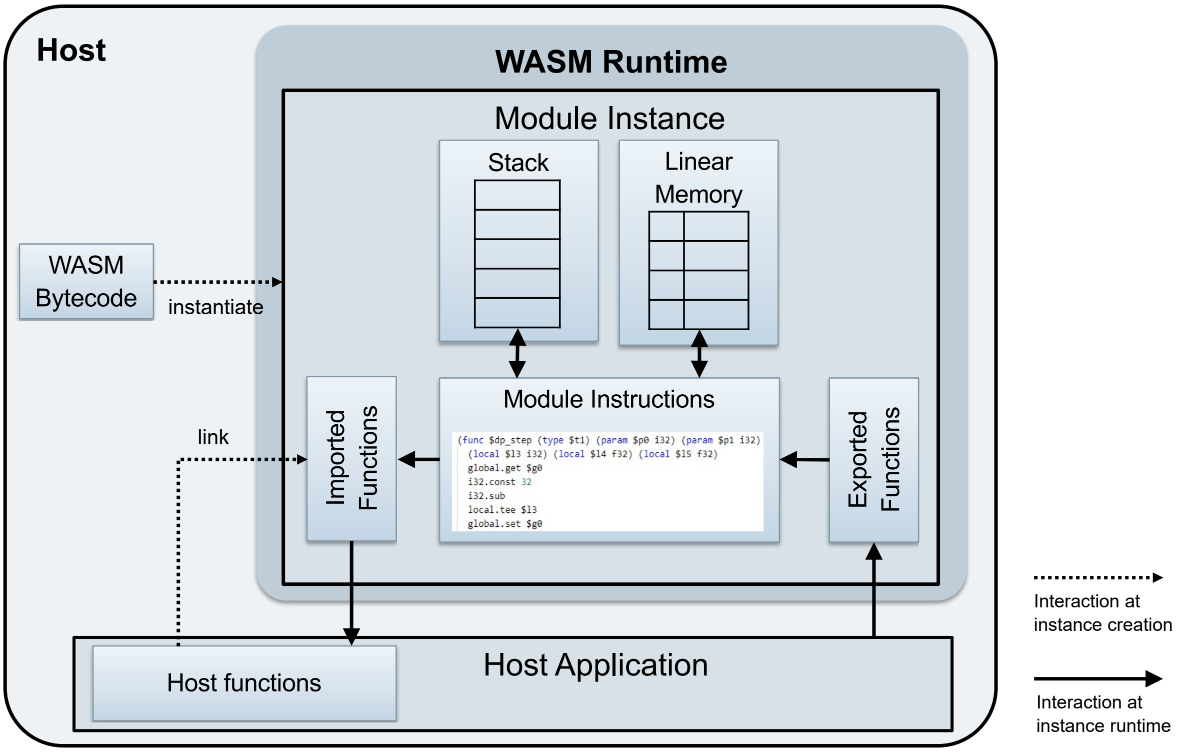
- The bytecode is portable and can be executed on any hardware for which a Wasm runtime exists.
- Performance can be close to that of native machine code, at least in environments where just-in-time (JIT) or ahead-of-time (AOT) compilation to native machine instructions is possible.
- Module instances are sandboxed and cannot interfere with each other or the host environment in uncontrolled ways.
- Implementation of modules can be carried out in any programming language that supports compilation to Wasm.
- Module code can be statically analyzed for memory usage and instruction counts.
- The runtime has complete control over the execution such that running bytecode instances can be preempted and a resumable snapshot of an instance's state can be taken by the runtime without requiring explicit support by the module implementation.
- There is no undefined behavior, all operations are deterministic and hence the computed outputs are identical across different Wasm runtimes and host environments.

As downsides of this approach the overhead due to the WebAssembly runtime needs to be mentioned (unless the modules are AOT compiled, which is only possible on limited set of platforms, and adds an additional compilation step before execution). On platforms where JIT or AOT compilation is not available (e.g., on many embedded devices), Wasm modules need to be interpreted which causes substantial performance degradation.

A WebAssembly module may interact with its host environment through imported and exported functions. Function imports are declared by name and signature and linked by the runtime when the module is instantiated. Function exports are also declared by name and signature and can be called by the runtime once the module is instantiated. A further mechanism for exchanging data is the use of the linear memory blocks. A schematic overview of the interactions between a Wasm module and its host is shown in 2.

3.2 Interface

According to the description of executable simulation modules given in Section 1, the interface that the module



```
(func $dp_step (type $t1) (param $p0 i32) (param $p1 i32)
  (local $l3 i32) (local $l4 f32) (local $l5 f32)
  global.get $g0
  i32.const 32
  i32.sub
  local.tee $l3
  global.set $g0
```

Figure 2. Interactions between WebAssembly modules and the host.

exposes includes the following functionality:

- Retrieve a model description from the module specifying input, state and output variables, module capabilities as well as potential constraints on compatible timestep lengths,
- create a new instance of the simulation module and initialize it with given parameters and initial values,
- set values for the input variables,
- calculate forward in time by a given timestep,
- retrieve the resulting values of the output and state variables,
- reset state to the previous timestep,
- terminate the instance.

The creation of such high-level interfaces between WebAssembly modules and their host is complicated by the fact that core WebAssembly only knows four basic data types (32bit and 64bit variants of integers and floating point numbers), and the signatures of any declared export functions need to be expressed in terms of these data types. More complex data structures can only be exchanged by serializing them to linear memory, which is accessible both to the module instance and to the host. Then, pointers to locations inside linear memory can be passed as arguments to a regular Wasm exported functions (pointers into linear memory are just offsets from the start of the memory block). To handle heap allocation inside the linear memory in a consistent way, a *dtasm* module exports an allocator and a corresponding de-allocator function.

Since serialization and de-serialization need to be performed on either side of the Wasm sandbox for this to work, a serialization format should be chosen that is not just lightweight, but also has implementations in many different programming languages. After careful consideration, we picked the FlatBuffers ((FlatBuffers 2021)) serialization library for the *dtasm* ABI. In FlatBuffers, data structures are described by a schema written in an interface definition language (IDL), and a compiler provided by the FlatBuffers project then generates source code for serialization/de-serialization of such structures for a variety of target languages. The generated code is very performant and lightweight (e.g., the code generated for C++ is a single, self-contained header file), and code generation supports a wide range of contemporary programming languages. Furthermore, FlatBuffers (at least in some programming languages) allows the validation of binary buffers for a given schema. This is an important feature for enforcing the security boundary between host and modules and for increasing the robustness of implementations.

The sequence of events for invoking a *dtasm* interface function generally follows these steps:

- The host assembles the input data into a FlatBuffer, determines its size and invokes the allocator function exported by the *dtasm* module instance to allocate a buffer in linear memory of the instance.
- The host serializes the input FlatBuffer into the allocated memory block.
- The host allocates an additional buffer (of a default size) for holding the results of the call.
- The host invokes the interface function, passing pointers to the in- and output buffers as well as their

respective sizes.
- The module decodes the input buffer, processes it and assembles the result into a result FlatBuffer.
- If the output buffer is sufficiently large to hold the result FlatBuffer, the result is written to the output buffer and the call returns. If not, the size of the result is returned, the host allocates a new large enough output buffer and invokes the interface function again with the same input.
- The host reads the result FlatBuffer from the output buffer.

For a given programming language, much of the logic involved in this procedure can be encapsulated into auxiliary libraries, such that the consumers of the ABI don't need to deal with the low-level details.

3.3 Model Description

Similar to the model description defined by the FMI standard, a *dtasm* module provides a model description that contains

- metadata about the module (name, id, when and by what tool it was created),
- module capabilities (e.g., can timesteps be reset, can the module utilize derivative information),
- a list of model variables together with their causality, data type and default value,
- infos about valid experiment conditions such as constraints on compatible timestep size, start and end time of simulations.

Model variables can be of causality parameter, input, local or output, and the supported data types are real, integer, boolean and string.

A more detailed description can be found as part of the code repository in (dtasm 2021). The binary format of the model description is again described by a FlatBuffers schema. Since FlatBuffers supports creation of buffers from JSON files that are compatible with the schema, model descriptions can be authored using JSON for convenience. The binary representation is then embedded into the module as a byte array literal, and can be retrieved from instances of the module by invoking the `getModelDescription` interface function.

Since module instantiation is already a part of the WebAssembly specification, no explicit interface function is needed for instantiation. Likewise, since WebAssembly modules cannot use any native resources, an explicit `terminate` function in the interface is not needed and instances can be terminated and disposed of simply by unloading them from the WebAssembly runtime.

4 Features and Limitations of *dtasm*

4.1 Features

Using WebAssembly bytecode as the target format for executable simulation modules has some interesting implications that we discuss in the following. As WebAssembly is a very simple bytecode format, it is easy to target by compilers for high-level programming languages, which is confirmed by the number of existing compilers supporting Wasm as output. On first look bytecode seems like an unusual format for executable numerical code. But considering, e.g., the LLVM compiler architecture (Lattner and Adve 2004), it is based on a separation between frontend and backend compilation, where the frontend generates intermediate bytecode (LLVM Intermediate Representation (IR)) that is compiled to native machine code by the backend. WebAssembly can be thought of as replacing the intermediate bytecode by a portable, well-specified format that can be easily targeted by other compilers as well. The development of tools, infrastructure and supporting standards around WebAssembly has been strongly driven by the Web community during recent years, which has lead to a number of high-quality implementations and standards being available as open source (e.g. Zakai (2011), WASI (2021), and AssemblyScript (2021)).

Instances of WebAssembly modules can store internal state on the stack, in linear memory or in global variables (but as globals are seldomly used for this purpose, we disregard them here). When a module instance is not currently executing a function, its stack is empty, so that a snapshot of its state can be created simply by dumping the content of its linear memory (which is just a contiguous byte array) to a file. The instance can then be terminated, a new instance be created and its linear memory read back from the file, and the new instance then has exactly the same internal state as the previous one. All this can be achieved solely from the runtime without any explicit support by the module implementation. The memory dump is even portable across different WebAssembly runtimes, as the mechanism of linear memory is specified by the WebAssembly standard. E.g., this method could also be used to reset timesteps for modules that do not explicitly support such functionality:

1. Store a dump of the linear memory after each timestep.
2. If a timestep needs to be reset, the previous dump is loaded into the instance's memory to reset its state.

Depending on the size of the module's memory, this procedure can be quite expensive, hence explicit timestep resetting support by the module should be preferred when available.

Some more advanced features could even include preemption of running module instances by the runtime, relocation to other machines and resumption at the exact state where preemption happened. Such operations are not yet widely supported by popular Wasm runtimes, but many projects are rapidly adding features in this direction. Preemptive multitasking could prevent individual module instances from occupying computational resources and allow a fair distribution of resources to all running instances. Related is the concept of gas counting: The runtime can monitor the consumed instruction count ("gas") of a WebAssembly function and preempt the instance if the instruction count exceeds a certain threshold. This could allow a fair distribution of compute resources among multiple active module instances.

4.2 Limitations

The WebAssembly standard in its current stage has some limitations that impact its usefulness as a packaging format for executable simulations:

- Not all features of low-level programming languages can be mapped cleanly to WebAssembly. In particular, non-local jumps, stack unwinding or multithreading do not currently have support in WebAssembly, although extensions of the standard for supporting these features are planned.
- Specialized hardware acceleration units like GPUs or TPUs are not accessible to WebAssembly modules. Support would need customized implementations outside of the Wasm specification.
- WebAssembly modules are statically linked, which makes them rather large in size (an extension of the Wasm spec allowing dynamic linking is planned).
- Available development tooling, especially in regards to debugging support, is lacking behind other more established ecosystems.
- The size of linear memory blocks is given in multiples of 64kB, which is wasteful on embedded platforms.
- Security of the Wasm sandbox model is not perfect, e.g., side-channel attacks are not prevented by the specification but need to be mitigated by runtime individually.

Also, the general overhead of a WebAssembly runtime in terms of performance and memory usage is certainly not negligible. Especially on embedded platforms, AOT compilation or JIT compilation are often not available or not feasible, so the only option are interpreters that are generally an order of magnitude slower than native code (see Wasm3 (2021)). Very small devices with less than 64kB memory or no support for dynamic memory allocation are not suitable for running *dtasm* modules. Performance and size of the runtime is often a tradeoff: While interpreters can be very lightweight (Wasm3 is around 100kB in size when compiled), JIT runtimes on the other hand include native code generators and thus are often several tens of megabytes in size.

5 Prototypical Implementation

Several implementations of *dtasm* runtimes and modules have been developed during the course of this work, some of which are available as open source (dtasm 2021).

5.1 Runtimes

Dtasmtime is a *dtasm* runtime library implemented in Rust that builds upon (Wasmtime 2021), a popular open source engine for WebAssembly modules featuring JIT compilation. *Dtasmtime* supports loading and execution of *dtasm* modules as well as saving and loading of instance state to and from files. Interaction with *dtasmtime* from Rust applications happens through a high-level API, while an additional lower-level C-compatible API is provided in order to facilitate integration of the library into C/C++ and other programming languages.

Additional *dtasm* runtimes have been implemented based on the Wasm3 interpreter (Wasm3 2021) and the V8 JavaScript engine. While Wasm3 by nature of interpretation is substantially slower in execution performance than JIT or AOT compiling runtimes, it is very lightweight and allows execution of *dtasm* modules on embedded targets such as Arduino-class microcontrollers (see Figure 3). Implementation of a *dtasm* runtime in JavaScript allowed running *dtasm* modules inside contemporary web browsers as well.

5.2 Modules

For demonstration and benchmark purposes, a simple double pendulum simulator (based on Wheatland (2004)) has been implemented in C++ and Rust, and compiled into a *dtasm* module using the WASI SDK (2021) in the case of C/C++ and Rust's integrated `wasm32-wasi` target. Source code for both versions is available (dtasm 2021).

During the course of our experimentation, we also compiled *dtasm* modules from several source FMUs created by various commercial and open source simulation tools (Simulink/FMIKit, Dymola, OpenModelica). While most of the resulting *dtasm* modules could be successfully compiled and executed, some FMUs were found to utilize C/C++ functionality for error handling (e.g., exceptions, non-local jumps) that are currently not supported by WebAssembly and had to be stubbed in order to compile successfully. Such *dtasm* modules were then only operable under non-error conditions. As some FMUs utilized resource files that are read at runtime (which core WebAssembly does not support), the WASI interface for reading files from the host file system had to be made available when executing such *dtasm* modules. Accessing external files violates the self-containedness assumption on *dtasm* modules and also may have security implications. If a direct *dtasm* export was integrated into such simulation tools (without the detour through FMU), this issue could be avoided by embedding additional resource files directly into the WebAssembly module.

5.3 Performance

One of the most interesting questions regarding *dtasm* is the resulting performance overhead when comparing to execution of native machine code, since this is one of the major tradeoffs incurred by *dtasm*. This overhead consists of several distinct contributions:

1. Raw computational performance of WebAssembly compared to native machine code,
2. overhead afforded by the module interface, mainly through copying of memory blocks and

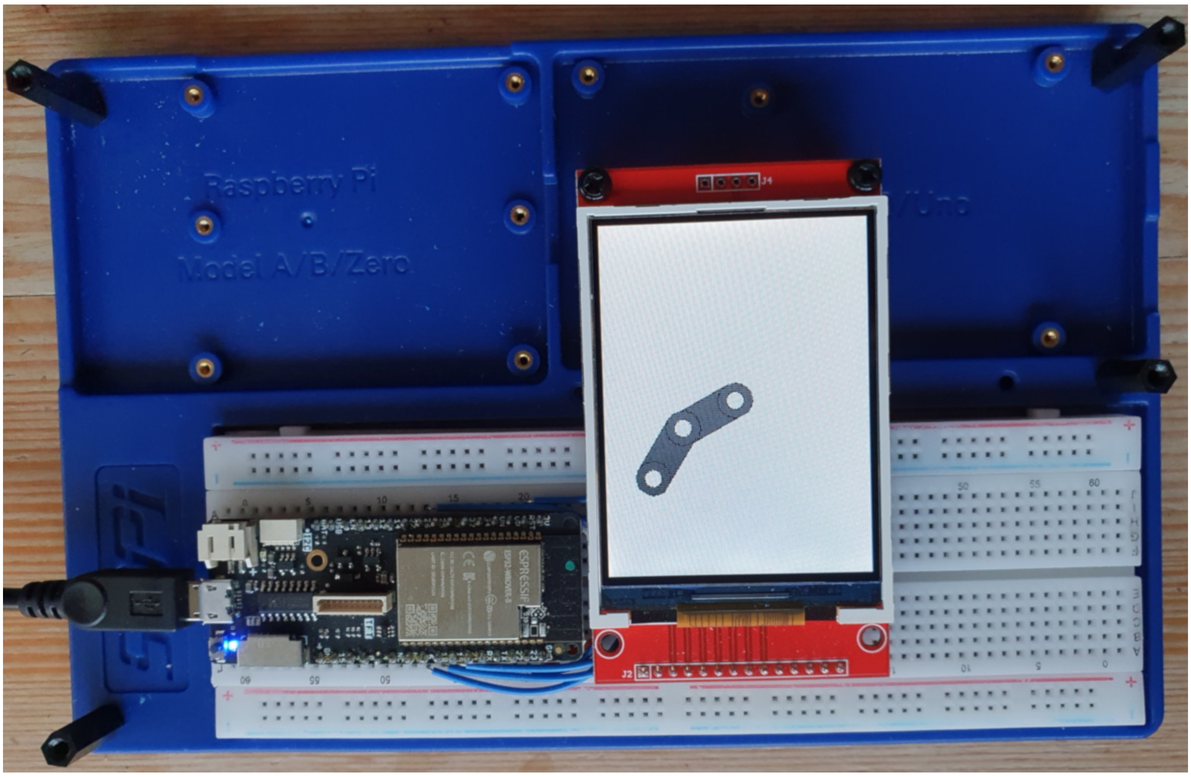


Figure 3. Double pendulum simulator generated from a Simulink model running as *dtasm* module on an ESP32 MCU.

serialization/de-serialization,

3. performance overhead due to how efficiently the Wasm runtime implements calls to Wasm export functions and access to the Wasm linear memory,
4. optimization capabilities of the compiler used to create the native machine code and the Wasm module respectively.

Characterization of these overheads in isolation is difficult and will not be attempted here. The raw performance overhead of Wasm for different engine implementations has been the subject of many benchmarks (see, e.g., Jangda et al. (2019), Denis (2021), and Wasm3 (2021)), in which a best-case factor for JIT-based engines between 1.5 and 2.5 has been found, depending on the workload and Wasm engine considered.

To compare performance of the *dtasm* prototype implementation to native execution, we used the C++ source code of our double pendulum module and added an outer loop that runs the simulation for a fixed number of steps, still using the *dtasm* interface but directly from C++. This combination was then compiled to native machine code using GNU compiler collection (gcc). We performed the same computation using the LLVM-compiled *dtasm* module running in *dtasmtime*, and compared execution times. Figure 4 shows the result for 10 million time steps of the double pendulum simulator. The overhead of *dtasmtime* is found to be around a factor of 2.4.

In an attempt to reduce the influence of 2 and 3 above (in particular the overhead incurred by the *dtasm* interface), we adapt the implementation of the double pendulum simulator to internally perform many small time steps of fixed size, and reduce the number of steps on the outer loop, thereby invoking the *dtasm* interface less often than in the first case. Figure 5 shows the results for 10^5 inner time steps and 10^4 outer loop steps (amounting to 10^9 time steps total).

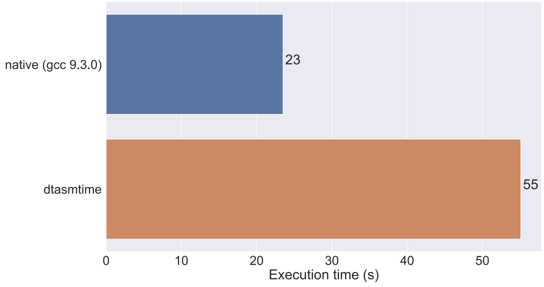


Figure 4. Execution times of the double pendulum simulator for 10^7 time steps.

It can be seen that native and *dtasmtime* performance are almost identical in this case. This implies that most of the overhead incurred by running simulations as *dtasm* modules indeed is due to interface calls. We note that the significance of this comparison is very limited though, because we only tested a single simulation module. Many features of more realistic simulators, such as extensive numerical linear algebra operations, may yield a different picture here. Also, no significant code optimizations have been applied to either the simulation module or the

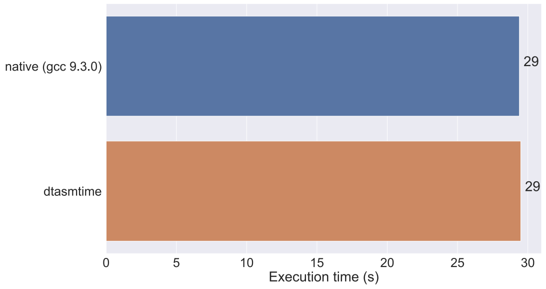


Figure 5. Execution times of the double pendulum simulator for 10^5 inner time steps and 10^4 outer time steps.

dtasmtime implementation.

The performance of Wasm interpreters such as Wasm3 was found to be around a factor of 10 slower compared to *dtasmtime*. Hence, the performance on embedded devices, where JIT compilation is not an option due to resource constraints, must be expected much lower than what is reported above. While AOT compilation to native code may be another option for such targets, it hinges on the availability of suitable compilers.

6 Conclusion

Packaging executable simulation modules as native machine code poses several challenges related to portability and security: Machine code targets specific hardware platforms and is difficult to sandbox from its execution environment. Bytecode formats can help address both of these issues since they target abstract machines with enforceable security boundaries. Bytecode formats can serve as compilation targets for higher level programming languages, and application level virtual machines for bytecode often support secure sandboxes by design. WebAssembly in particular is suitable for executable simulation modules as it focuses on performance and is sufficiently low-level to be used as compilation target for many of the programming languages typically in use by numerical codes.

In this text, we introduced an efficient and language-independent interface to WebAssembly modules that in functionality resembles FMI for co-simulation. We discussed the rationale for our design decisions as well as the advantages and drawbacks they entail. The design is sufficiently lean to allow targeting embedded devices, although the overhead created by the need for a virtual machine is certainly considerable there.

We demonstrated feasibility by providing prototypical implementations of *dtasm* runtimes and modules. A preliminary performance test shows that the main overhead is due to the module interface (which is not specific to WebAssembly), but the performance of Wasm itself can be expected comparable to the performance of native binaries. Simulation code generated by established system simulation tools can often be compiled into *dtasm* modules with manageable effort, allowing *dtasm* to take advantage of the existing system simulation ecosystem, e.g., through the export of source FMUs that are then compiled into *dtasm* modules.

Dtasm modules can be instrumented at runtime in a way that allows dynamic re-allocation to other compute nodes at runtime. In the future, this could enable orchestration systems that dynamically dispatch running module instances to compute nodes according to available resources. Compute nodes close to the shop floor could then be utilized as a single cluster instead of individually configured devices.

While WebAssembly is still a comparably young technology, it has beneficial properties regarding portability as well as sandboxing and shows promising results regarding performance. It remains to be seen if WebAssembly can be a relevant technology for packaging numerical simulations in the future. A further adoption would certainly hinge on support by existing system simulation tools to export *dtasm* modules. Using source FMUs as an intermediary for compiling to WebAssembly could be a viable path forward in this direction.

References

AssemblyScript (2021). “A language made for WebAssembly”. URL: https://www.assemblyscript.org/ (visited on 2021-03-14).

Blochwitz, Torsten, Martin Otter, J. Akesson, et al. (2012). “Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models”. In: *9th International Modelica Conference*. URL: https://elib.dlr.de/78486/.

Blochwitz, Torsten, Martin Otter, Martin Arnold, et al. (2011-03). “The Functional Mockup Interface for Tool independent Exchange of Simulation Models”. In: *8th International Modelica Conference*. Ed. by Christoph Clauß. Linköping Electronic Conference Proceedings. Linköping University Press, pp. 105–114. URL: https://elib.dlr.de/74668/.

Boschert, Stefan, Christoph Heinrich, and Roland Rosen (2018). “Next generation digital twin”. In: *Proc. TMCE 2018*. Vol. 2018, pp. 7–11.

Denis, Frank (2021). “Benchmark of WebAssembly runtimes – 2021 Q1 edition”. URL: https://github.com/jedisct1/webassembly-benchmarks/tree/master/2021-Q1 (visited on 2021-03-13).

dtasm (2021). “Digital Twin Assembly - A portable and sandboxed package format for executable simulation modules based on WebAssembly”. URL: https://github.com/siemens/dtasm (visited on 2021-04-29).

FlatBuffers (2021). “An efficient cross platform serialization library”. URL: https://google.github.io/flatbuffers/ (visited on 2021-03-13).

Haas, Andreas et al. (2017). “Bringing the Web up to Speed with WebAssembly”. In: *SIGPLAN Not.* 52.6, pp. 185–200. DOI: 10.1145/3140587.3062363.

Hall, Adam and Umakishore Ramachandran (2019). “An execution model for serverless functions at the edge”. In: *Proceedings of the International Conference on Internet of Things Design and Implementation*, pp. 225–236.

Hatledal, Lars Ivar et al. (2019). “Fmu-proxy: A framework for distributed access to functional mock-up units”. In: *Pro-*

ceedings of the 13th International Modelica Conference. Linköping University Electronic Press.

Jacobsson, Martin and Jonas Willén (2018). "Virtual machine execution for wearables based on webassembly". In: *EAI International Conference on Body Area Networks*. Springer, pp. 381–389.

Jangda, Abhinav et al. (2019). "Not so fast: Analyzing the performance of webassembly vs. native code". In: *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pp. 107–120.

Lattner, Chris and Vikram Adve (2004). "LLVM: A compilation framework for lifelong program analysis & transformation". In: *International Symposium on Code Generation and Optimization, 2004. CGO 2004.* IEEE, pp. 75–86.

Peach, G. et al. (2020). "eWASM: Practical Software Fault Isolation for Reliable Embedded Devices". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11, pp. 3492–3505. DOI: 10.1109/TCAD.2020.3012647.

Rasheed, Adil, Omer San, and Trond Kvamsdal (2020). "Digital Twin: Values, Challenges and Enablers From a Modeling Perspective". In: *IEEE Access* 8, pp. 21980–22012.

Rossberg, Andreas (2019). "WebAssembly Core Specification". URL: https://www.w3.org/TR/2019/REC-wasm-core-1-20191205/ (visited on 2021-03-14).

Schijndel, A.W.M. (Jos) van (2014). "A review of the application of SimuLink S-functions to multi domain modelling and building simulation". In: *Journal of Building Performance Simulation* 7.3, pp. 165–178. DOI: 10.1080/19401493.2013.804122.

Tao, Fei and Meng Zhang (2017). "Digital Twin Shop-Floor: A New Shop-Floor Paradigm Towards Smart Manufacturing". In: *IEEE Access* 5, pp. 20418–20427.

WASI (2021). "The WebAssembly System Interface". URL: https://wasi.dev/ (visited on 2021-03-14).

WASI SDK (2021). "WASI-enabled WebAssembly C/C++ toolchain". URL: https://github.com/WebAssembly/wasi-sdk (visited on 2021-03-14).

Wasm3 (2021). "Performance". URL: https://github.com/wasm3/wasm3/blob/master/docs/Performance.md (visited on 2021-03-13).

Wasmtime (2021). "A small and efficient runtime for WebAssembly & WASI". URL: https://wasmtime.dev/ (visited on 2021-03-13).

Wheatland, Michael S. (2004). "The Double Pendulum". URL: http://www.physics.usyd.edu.au/~wheat/dpend_html/ (visited on 2021-03-14).

Zakai, Alon (2011). "Emscripten: an LLVM-to-JavaScript compiler". In: *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pp. 301–312.

Zheng, Gavin et al. (2021). "WebAssembly (WASM)". In: *Ethereum Smart Contract Development in Solidity*. Singapore: Springer, pp. 317–334. DOI: 10.1007/978-981-15-6218-1_11.

New Method to Perform Data Reconciliation with OpenModelica and ThermoSysPro

Daniel Bouskela[1] Audrey Jardin[1] Arunkumar Palanisamy[2]
Lennart Ochel[2] Adrian Pop[3]

[1]EDF Lab Chatou, France, {daniel.bouskela,audrey.jardin}@edf.fr
[2] RISE AB, Sweden, {arunkumar.palanisamy,lennart.ochel}@ri.se
[3] PELAB - Linköping University, Sweden, adrian.pop@liu.se

Abstract

Data reconciliation aims at improving the accuracy of measurements by reducing the effect of random errors in the data. This is achieved by introducing redundancies on the measured quantities in the form of constraints based on fundamental physical laws such as mass, momentum and energy balance equations. These constraints are called the auxiliary conditions. Modelica is an equational language that was conceived to express models based on first principle physics for the purpose of behavioral simulation. This paper shows how to reuse such models for the purpose of data reconciliation. The novelty is to automatically extract the auxiliary conditions from the Modelica model. Then the reconciled values are computed using a least square method constrained by the auxiliary conditions, as specified by the VDI 2048 standard. The new method has been implemented in OpenModelica. A simple example built with ThermoSysPro illustrates the method in detail.

Keywords: data reconciliation, Modelica, model reuse, cyber-physical systems, structural analysis

1 Introduction

The safe and efficient operation and maintenance of power plants rely on plant data. Therefore, ensuring the quality of plant measurements such as pressures, temperatures and mass flow rates is essential. However, plant data are subject to measurement errors that put a limitation on their efficient use for plant monitoring, diagnosis and prognosis because they lead to uncertainties in the assessment of the plant state. The consequence is a decrease in production because of safety regulations that put stringent limits on plant operation. It is therefore important to compute the best estimates of the measurement uncertainties in order to regain satisfactory operational margins. Best estimates can be obtained by combining statistics on the data with a priori knowledge from the expert expressed in the form of physical models. Data reconciliation has been conceived for the process industry with this principle in mind and is the subject of the VDI 2048 standard (VDI, 2017). It has been used for several process related issues such as finding lost megawatts in power plants (Langenstein et al., 2004) or detecting sensor and actuator faults in hydraulic systems (Bedjaoui et al., 2008). This paper follows the VDI 2048 standard methodology.

Data reconciliation aims at improving the accuracy of measurements by reducing the effect of random errors in the data. The main difference between data reconciliation and other data improvements techniques is that data reconciliation uses a model to express the physical constraints on the variables of interest and adjusts their measured values such that the estimates satisfy the constraints: the variables are thus reconciled. The physical constraints on the variables of interest are called the *auxiliary conditions*. The main benefit of introducing redundancy in the form of auxiliary conditions is that the estimates have lower uncertainties than the initial measurements.

Dedicated tools such as VALI (Belsim, 2021) exist to perform data reconciliation, but they require the physical models to be specifically developed for that purpose. This makes data reconciliation costly and difficult to use. A natural answer to that problem is to perform data reconciliation on existing Modelica models (Modelica Association, 2021), developed and validated for the general purpose of plant operation at large.

The objective of this paper is to present a new method to perform data reconciliation using Modelica models. The novelty of the method lies in the automatic extraction of the auxiliary conditions from the Modelica model. Once the auxiliary conditions are extracted, the variables of interest can be reconciled using the inputs provided by the user in the form of measurement data and correlation matrices, and the numerical procedure described in the VDI 2048 standard.

Although dynamic data reconciliation is possible (Bedjaoui et al., 2008; Bai and Thibault, 2010), the VDI 2048 standard considers only *steady-state data reconciliation*, which means that measurements are conducted while the system is under quasi steady-state conditions. For the parts of the system where this assumption is not valid, additional uncertainties must be added to fluctuating quantities. Therefore, the new method presented in this paper only applies to static models, i.e. models with physical laws expressed in the form of algebraic equations. These models can be obtained by removing the differential part of the physical equations. Also, the VDI 2048 assumes that the auxiliary conditions are *exact physical laws*. This means that the model should be sufficiently representative of the real system, for instance that leaks, or other serious disturbances, are not overlooked. For the quantities such as temperatures that cannot be represented by exact physical laws, e.g. when dealing with thermal correlations, additional uncertainties must be added to the quantities to account for the uncertainties in the physical laws. The uncertainties on variables of interest that cannot be directly measured such as specific enthalpies can be obtained using uncertainty propagation techniques (Dutfoy et al., 2009).

The data reconciliation method is summarized in Section 2. The new algorithm to automatically extract the auxiliary conditions from a Modelica model is given and applied to perform data reconciliation on the simple example of a splitter in Section 3.

2 Data Reconciliation in a Nutshell

2.1 Mathematical Formulation

Let $\hat{x} = \{\hat{x}_i\}_{1 \le i \le n}$ be measurements on physical quantities $x = \{x_i\}_{1 \le i \le n}$ that are constrained by exact physical laws represented by a set of algebraic equations $\mathcal{C}$ such that

$$\mathcal{C}(\bar{x}) = 0 \tag{1}$$

where $\bar{x}$ denotes the vector of the true values of x. The true values are unknown, and the objective is to provide the best estimates for them. Notice that in general the measurements do not satisfy Equation (1): $\mathcal{C}(\hat{x}) \neq 0$.

The random unbiased error ε_i on measurement $\hat{x}_i$ is described by a Gaussian noise around $\bar{x}_i$:

$$\hat{x}_i = \bar{x}_i + \varepsilon_i \tag{2}$$

with

$$\varepsilon_i = \mathcal{N}(0, \sigma^2) \tag{3}$$

$\mathcal{N}(0, \sigma^2)$ being a Gaussian distribution of mean value 0 and standard deviation σ. This is valid according to the central limit theorem if a sufficient number of measurements are conducted with sufficient accuracy. The mean value of Equation (2) yields the true value of $\hat{x}_i$:

$$\bar{x}_i = E(\hat{x}_i) \tag{4}$$

However, computing the true value using Equation (4) requires a sufficient number of measurements with different sensors and measuring techniques to avoid biases, which is impractical when dealing with a large number of measured quantities.

To find an estimate of the true value, the VDI 2048 standard states that the Gaussian distribution is applicable, even with only one measured value for each variable x_i. This assertion is justified by the fact that each measured value deviates from the true value by a sum of random, mostly independent deviations that makes the central limit theorem applicable with good approximation.

The weight or half-width confidence interval for $\hat{x}_i$ is defined by:

$$w_{\hat{x},i} = \lambda_p \cdot \sigma_{\hat{x},i} \tag{5}$$

where $\sigma_{\hat{x},i} = \sqrt{E[(\hat{x}_i - \bar{x}_i)^2]}$ is the standard deviation of $\hat{x}_i$ and λ_p is the quantile of normal distribution with probability p. Then $\bar{x}_i$ lies within the confidence interval $\hat{x}_i \pm \lambda_p \cdot \sigma_{\hat{x},i}$ with the probability p:

$$P\left(|\bar{x}_i - \hat{x}_i| \le \lambda_p \cdot \sigma_{\hat{x},i}\right) = p \tag{6}$$

For $p = 95\%$, $\lambda_p = 1.96$ which yields the following confidence interval:

$$w_{\hat{x}_i} = \lambda_{95\%} \cdot \sigma_{\hat{x},i} = 1.96 \cdot \sigma_{\hat{x},i} \tag{7}$$

The covariance matrix of $\hat{x}$ is defined as:

$$S_{\hat{x}} = \begin{cases} S_{\hat{x},i,i} = {\sigma_{\hat{x},i}}^2 \\ S_{\hat{x},i,j} = r_{\hat{x},i,j} \cdot \sigma_{\hat{x},i} \cdot \sigma_{\hat{x},j} \end{cases} \tag{8}$$

where the correlation coefficients $r_{\hat{x},i,j}$ are such that $\left|r_{\hat{x},i,j}\right| \le 1$. $r_{\hat{x}}$ is the correlation matrix. Expressed as function of the weights, the covariance matrix is:

$$S_{\hat{x}} = \begin{cases} S_{\hat{x},i,i} = \left(w_{\hat{x},i}/\lambda_p\right)^2 \\ S_{\hat{x},i,j} = r_{\hat{x},i,j} \cdot w_{\hat{x},i} \cdot w_{\hat{x},j}/\lambda_p^2 \end{cases} \tag{9}$$

The estimated values are found in the form of the reconciled values of x which are denoted $\bar{\bar{x}}$ in the sequel. They are obtained by finding the point in the subspace defined by Equation (1) which is closer to measurements with lower uncertainties than measurements with higher uncertainties. They are thus computed using a least square method where the weighing matrix is the inverse of the covariance matrix. Therefore, the objective function is:

$$J(x) = (x - \hat{x}) \cdot S_{\hat{x}}^{-1} \cdot (x - \hat{x}) \tag{10}$$

and the minimization problem to be solved is:

$$\begin{cases} J(\bar{\bar{x}}) = \min_x \left((x - \hat{x}) \cdot S_{\hat{x}}^{-1} \cdot (x - \hat{x})\right) \\ \mathcal{C}(\bar{\bar{x}}) = 0 \end{cases} \tag{11}$$

The vector of improvements is defined as the difference between the estimated values and the measured values:

$$\nu = \bar{\bar{x}} - \hat{x} \tag{12}$$

The covariance matrices of the reconciled values and of the improvements, derived from the general formula of error propagation, are respectively:

$$S_{\bar{\bar{x}}} = \frac{\partial \bar{\bar{x}}}{\partial \hat{x}} \cdot S_{\hat{x}} \cdot \left(\frac{\partial \bar{\bar{x}}}{\partial \hat{x}}\right)^T \tag{13}$$

$$S_\nu = \frac{\partial \nu}{\partial \hat{x}} \cdot S_{\hat{x}} \cdot \left(\frac{\partial \nu}{\partial \hat{x}}\right)^T \tag{14}$$

Noticing that

$$D_{S_{\hat{x}}^{-1}}(\nu_1, \nu_2) = \nu_1 \cdot S_{\hat{x}}^{-1} \cdot \nu_2 \tag{15}$$

is a scalar product, then

$$J(x) = D_{S_{\hat{x}}^{-1}}(x - \hat{x}, x - \hat{x}) = \|x - \hat{x}\|^2_{S_{\hat{x}}^{-1}} \tag{16}$$

is the square of the distance between x and $\hat{x}$ weighted by $S_{\hat{x}}^{-1}$. Let r be the number of auxiliary conditions (i.e. the size of $\mathcal{C}$), which is also the redundancy level. Solving the minimization problem of Equation (11) amounts to finding a point $\bar{\bar{x}}$ on the r-dimensional surface defined by $\mathcal{C}(x) = 0$ which is the orthogonal projection of the measured point $\hat{x}$ on the surface $\mathcal{C}(x) = 0$. The projection is done according to a metrics such that the coordinates of the projected point are closer to measurements with lower uncertainties than to measurements with higher uncertainties. The uncertainties of the reconciled values are thus reduced because the distance from the true values to the orthogonal projection is always smaller than the distance to the initial point:

$$\|\bar{\bar{x}} - \bar{x}\|^2_{S_{\hat{x}}^{-1}} \le \|\hat{x} - \bar{x}\|^2_{S_{\hat{x}}^{-1}} \tag{17}$$

Moreover, the higher the value of r, the higher the uncertainty reduction because each time r is increased by one unit, one additional orthogonal projection is performed to a smaller subspace that brings the reconciled values closer to the true values. Thus, data reconciliation provides the best estimate of the variables of interest from the measured values $\hat{x}_i$, the weights $w_{\hat{x}_i}$ and the auxiliary conditions $\mathcal{C}(\bar{x}) = 0$.

The optimization problem of Equation (11) can be solved using the Lagrange multiplier method. The Lagrangian is:

$$L(x, \lambda) = J(x) + 2 \cdot \lambda^T \cdot \mathcal{C}(x) \tag{18}$$

where λ are the Lagrange multipliers. The values of x that yield the minimum value of $J(x)$ are obtained by solving the following equation system:

$$\begin{cases} \dfrac{\partial L}{\partial x} = 2 \cdot S_{\hat{x}}^{-1} \cdot\ (x - \hat{x}) + 2 \cdot \dfrac{d\mathcal{C}}{dx}^T \cdot \lambda = 0 \\ \dfrac{\partial L}{\partial \lambda} = \mathcal{C}(x) = 0 \end{cases} \tag{19}$$

The vector of contradictions is defined as:

$$u = (\hat{x} - \bar{x}) - (\bar{\bar{x}} - \bar{x}) = \hat{x} - \bar{\bar{x}} \tag{20}$$

Therefore, the vector of contradictions corresponds, in absolute value, to the vector of improvements. Its square value is:

$$J_0 = \|u\|^2_{S_{\hat{x}}^{-1}} = J(\bar{\bar{x}}) = u \cdot S_{\hat{x}}^{-1} \cdot u \tag{21}$$

Because u is standard normally (i.e. $\mathcal{N}(0,1)$) distributed, J_0 being the square of u is a χ^2-distributed function of r degrees of freedom (according to VDI 2048). Therefore, the following relationship holds with statistical certainty of probability p:

$$J_0 \le \chi^2_{r,p} \tag{22}$$

If Condition (22) is not satisfied, then the result for the reconciled values should be rejected because the vector of contradictions is too large. This can happen if some improvements are too large making the corresponding reconciled values fall out of their confidence ranges. This can be checked with the following individual tests:

$$|\bar{\bar{x}}_i - \hat{x}_i| / \sqrt{S_{v,i,i}} \le \lambda_p \tag{23}$$

where $S_{v,i,i}$ is the i^{th} diagonal element of the covariance matrix of the improvements. From the physical viewpoint, the failure of Condition (23) means that the constraints are not fully representative of the actual system behavior (e.g., some system leaks are not modelled), thus that the assumption that the constraints are exact physical laws is not verified, or that the measurements are incorrect (e.g., due to faulty sensors or poor estimations of their confidence level).

2.2 Numerical Resolution

The VDI 2048 standard recommends linearizing Equation (1) under the assumption that the improvements v are small:

$$\mathcal{C}(x) = \mathcal{C}(\hat{x}) + \frac{d\mathcal{C}}{dx}(\hat{x}) \cdot (x - \hat{x}) \tag{24}$$

and use an iterative method to solve Equation (19). This amounts to constructing the suite $\{x_k\}_{0 \le k \le N}$ such that:

$$\begin{cases} x_0 = \hat{x} \\ S_{\hat{x}}^{-1} \cdot (x_{k+1} - x_k) + \dfrac{d\mathcal{C}^T}{dx}(x_k) \cdot \lambda_k = 0 \\ \mathcal{C}(x_k) + \dfrac{d\mathcal{C}}{dx}(x_k) \cdot (x_{k+1} - x_k) = 0 \\ \bar{\bar{x}} = x_N \end{cases} \tag{25}$$

N is chosen to satisfy the convergence criteria:

$$L(x_N, \lambda_N)/r < \varepsilon \tag{26}$$

ε being a small number such as $\varepsilon = 10^{-10}$, and r being the size of $\mathcal{C}$.

Equation (25) can be rewritten as follows:

$$\begin{cases} x_0 = \hat{x} \\ x_{k+1} = x_k - S_{\hat{x}} \cdot F_k^T \cdot \lambda_k \\ F_k \cdot S_{\hat{x}} \cdot F_k^T \cdot \lambda_k = \mathcal{C}(x_k) \\ \bar{\bar{x}} = x_N \end{cases} \tag{27}$$

with

$$F_k = \frac{d\mathcal{C}}{dx}(x_k) \tag{28}$$

From Equation (19) and Equation (24), the reconciled values are:

$$\bar{\bar{x}} = \hat{x} - S_{\hat{x}} \cdot F_N^T \cdot (F_N \cdot S_{\hat{x}} \cdot F_N^T)^{-1} \cdot \mathcal{C}(\hat{x}) \tag{29}$$

From equations (13), (14) and (29), the correlation matrices of the improvements and of the reconciled values are given by:

$$S_v = S_{\hat{x}} \cdot F_N^T \cdot F^* \tag{30}$$

$$S_{\bar{\bar{x}}} = S_{\hat{x}} - S_v \tag{31}$$

where F^* is the solution of the equation

$$(F_N \cdot S_{\hat{x}} \cdot F_N^T) \cdot F^* = F_N \cdot S_{\hat{x}} \tag{32}$$

3 Performing Data Reconciliation with Modelica Models

3.1 Physical Laws and Boundary Conditions

A valid Modelica model is always a square model. A square model has as many unknown variables as equations to compute them. The model equations can be divided into two groups:

- The group of physical equations that represent physical laws such as the mass, momentum or energy balance equations, or empirical laws such as thermal or pressure losses correlations. This group is always underdetermined because physical laws express constraints between physical quantities, but do not provide any means to compute them in a unique way.
- The group of boundary conditions that provide the additional constraints to the group of physical equations to form a square system. Boundary conditions represent assumptions on the

environment of the system that are necessary to undertake a numerical experiment through simulation.

Let us take the Ohm's law as a simple example. The equation $U = R \cdot I$ expresses a constraint between the voltage U, the resistance R and the current I. To compute numerical values out of this equation, one must provide values for exactly two quantities in the form of boundary conditions, e.g., $U = 220\,V$ and $R = 50\,\Omega$, or provide a physical correlation to e.g. compute R as a function of the temperature T, in such case other boundary conditions related to the thermal condition of the system must be provided.

3.2 Well-posedness of the Data Reconciliation Problem

As boundary conditions do not represent physical laws, they cannot be part of the auxiliary conditions. Therefore, to use a valid Modelica model to represent the auxiliary conditions of a data reconciliation problem, the boundary conditions *that are related to the variables of interest* must be automatically removed from the Modelica model before computing the reconciled values. Each removed boundary condition must be replaced by a variable of interest in the sense explained in Section 3.6. It will be shown in the sequel that this action reduces by one unit the number of auxiliary conditions. When reducing the number of auxiliary conditions, it can happen that some variables of interest are not constrained anymore by any auxiliary condition.

Therefore, for the data reconciliation problem to be well-posed, the following conditions must be met:

$$f = n - r \geq 1 \tag{33}$$

$$r \geq 1 \tag{34}$$

$$\mathcal{C}(x_1, \ldots, x_n) = 0 \tag{35}$$

where f is the number of degrees of freedom, n is the number of variables of interest and r is the number of auxiliary conditions (or number of redundancies). Condition (35) means that all variables of interest must appear in at least one auxiliary condition. The variables of interest that do not appear in any auxiliary condition cannot be reconciled (which means that their reconciled values are equal to their measured values).

A valid Modelica model being a square model, $f = 0$ and Condition (33) is thus violated. This is another way to state why the model must be pre-processed to be fit for data reconciliation. The extraction algorithm presented in the sequel decreases the value of r by removing boundary conditions until no boundary conditions related to the variables of interest are left. Violation of Conditions (34) and (35) can thus happen when a group of variables of interest is related to a larger group of boundary conditions. The extraction algorithm will always satisfy Condition (33) because static models always involve boundary conditions, at least for energy systems as shown in (El Hefni and Bouskela, 2019).

3.3 Simple Example: Splitter

This section presents an illustrating example of a splitter that will be our companion throughout the rest of the paper.

A splitter is a device that separates an incoming flow into two outgoing flows. The physical laws of the splitter are the mass, momentum and energy balance equations. There is one mass balance equation and one energy balance equation to account for the flow separation, and three momentum balance equations, one for the incoming pipe and one for each outgoing pipe to account for the pressure losses inside the pipes. In the following it is assumed that all mass flow rates are positive, the fluid flowing from the left to the right in each pipe, cf. Figure 1.

The mass balance equation in the mixing volume is:

$$0 = Q_1 - Q_2 - Q_3$$

where Q_i is the mass flow rate of the fluid in the i^{th} pipe.

The energy balance equation in the mixing volume is:

$$0 = h_1 \cdot Q_1 - h_2 \cdot Q_2 - h_3 \cdot Q_3 + W$$

where h_i is the specific enthalpy of the fluid in the volume upstream of the i^{th} pipe, and W is the heating power.

The momentum balance equations in the 3 pipes are:

$$P_{1,l} - P_{1,r} = k_1.Q_1^2$$
$$P_{2,l} - P_{2,r} = k_2.Q_2^2$$
$$P_{3,l} - P_{3,r} = k_3.Q_3^2$$

where $P_{i,l}$ and $P_{i,r}$ are resp. the pressure of the fluid entering (at the left) and exiting (at the right) the i^{th} pipe (subscript *l* stands for *left* and subscript *r* stands for *right*), and k_i is the pressure loss coefficient in the i^{th} pipe, which is assumed to be an exact parameter.

The pressure P inside the mixing volume is related to the pressures in the neighboring pipes:

$$P = P_{1,r}$$
$$P = P_{2,l}$$
$$P = P_{3,l}$$

The specific enthalpies entering the outgoing pipes are equal to the specific enthalpy h in the mixing volume:

$$h_2 = h$$
$$h_3 = h$$

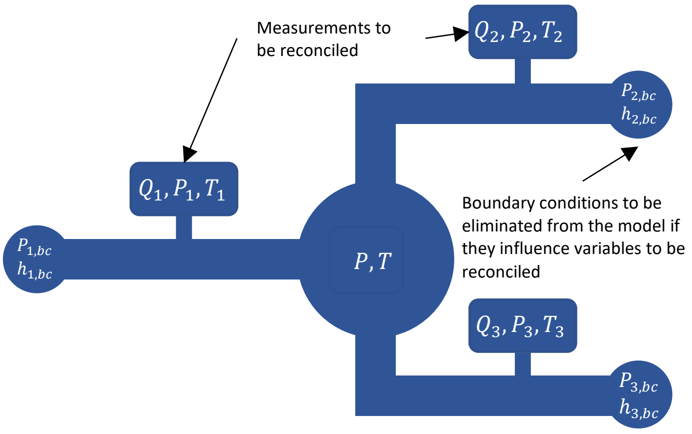

Measurements to be reconciled
Q_2, P_2, T_2
$P_{2,bc}$ $h_{2,bc}$
Q_1, P_1, T_1
$P_{1,bc}$ $h_{1,bc}$
P, T
Boundary conditions to be eliminated from the model if they influence variables to be reconciled
Q_3, P_3, T_3
$P_{3,bc}$ $h_{3,bc}$

Figure 1. Splitter

The mean pressures inside the pipes are given, by:

$$P_1 = \left(P_{1,l} + P_{1,r}\right)/2$$
$$P_2 = \left(P_{2,l} + P_{2,r}\right)/2$$
$$P_3 = \left(P_{3,l} + P_{3,r}\right)/2$$

where P_i is the mean pressure of the fluid in the i^{th} pipe.

The specific enthalpies in the volumes are related to the temperatures by the following equations:

$$h = c_p \cdot T + b \cdot P$$
$$h_1 = c_p \cdot T_1 + b \cdot P_1$$
$$h_2 = c_p \cdot T_2 + b \cdot P_2$$
$$h_3 = c_p \cdot T_3 + b \cdot P_3$$

where T_i is the mean temperature of the fluid in the i^{th} pipe. c_p is the specific heat capacity of the fluid and b accounts for the pressure dependence of the specific enthalpy. Those two last quantities are assumed to be exact parameters.

This model has 17 equations and 21 unknowns. Therefore 4 boundary conditions are needed. There are many different possibilities for choosing the boundary conditions. For instance, one may fix the pressure at each open end of the 3 pipes, or fix the pressure at the inlet of the incoming pipe and the mass flow rates inside the 2 outgoing pipes. Additionally, the specific enthalpy or the temperature of the fluid at the inlet of the incoming pipe must be fixed, so that 4 boundary conditions are properly fixed. The way to select the boundary conditions is important to define the proper scenarios to perform validation tests of the model, but does not matter for data reconciliation as they will be eliminated.

For the purpose of this example, the equations involving boundary conditions are:

$$P_{1,l} = P_{1,bc}$$
$$Q_2 = Q_{2,bc}$$
$$Q_3 = Q_{3,bc}$$
$$h_1 = h_{1,bc}$$

where $P_{1,bc}$, $Q_{2,bc}$, $Q_{3,bc}$ and $h_{1,bc}$ are the boundary conditions with fixed values.

3.4 The Set $\mathcal{C}$ of Auxiliary Conditions and the Set $\mathcal{S}$ of Intermediate Equations

The auxiliary conditions $\mathcal{C}(x) = 0$ must be automatically extracted from the Modelica model $\mathcal{M}(x, z) = 0$, where x are the variables to be reconciled, and z are the other variables of the model.

The equations in $\mathcal{C}(x) = 0$ almost always involve a subset y of z. This is why $\mathcal{C}(x) = 0$ will be denoted $\mathcal{C}(x, y) = 0$ in the sequel. The vector x will be called the *variables of interest* or the *known variables* as measurement values are provided for x. The vector y will be called the *intermediate variables*.

The extraction problem consists in extracting the set of auxiliary conditions $\mathcal{C}(x, y) = 0$ and the set $\mathcal{S}(x, y) = 0$ of intermediate equations that compute the intermediate variables from the known variables. Therefore, the system $\mathcal{C}(x, y) = 0$ is a non-square problem that has more variables of interest than equations, while $\mathcal{S}(x, y) = 0$ is a square system that has as many equations as intermediate variables.

3.5 Reformulating the Data Reconciliation Problem with Sets $\mathcal{C}$ and $\mathcal{S}$

Equation (27) is rewritten to reveal set $\mathcal{S}$:

$$\begin{cases} x_0 = \hat{x} \\ \mathcal{S}(x_0, y_0) = 0 \\ x_{k+1} = x_k - S_{\hat{x}} \cdot F_k^T \cdot \lambda_k \\ F_k \cdot S_{\hat{x}} \cdot F_k^T \cdot \lambda_k = \mathcal{C}(x_k) \\ \mathcal{S}(x_k, y_k) = 0 \\ \bar{\bar{x}} = x_N \end{cases} \tag{36}$$

with

$$F_k = \frac{d\mathcal{C}}{dx}(x_k, y_k) \tag{37}$$

F_k is computed by solving the following equation system:

$$\begin{cases} \dfrac{d\mathcal{C}}{dx} = \dfrac{\partial \mathcal{C}}{\partial x} + \dfrac{\partial \mathcal{C}}{\partial y} \cdot \dfrac{dy}{dx} \\ \dfrac{\partial \mathcal{S}}{\partial x} + \dfrac{\partial \mathcal{S}}{\partial y} \cdot \dfrac{dy}{dx} = 0 \end{cases} \tag{38}$$

The Jacobian matrices $\frac{\partial \mathcal{C}}{\partial x}, \frac{\partial \mathcal{C}}{\partial y}, \frac{\partial \mathcal{S}}{\partial x}$ and $\frac{\partial \mathcal{S}}{\partial y}$ can be computed analytically from sets $\mathcal{C}$ and $\mathcal{S}$.

3.6 Algorithm to Extract Set $\mathcal{C}$ and Set $\mathcal{S}$

The extraction algorithm relies on the BLT (Block Lower Triangular) decomposition of the equation system of the full Modelica model $\mathcal{M}$.

Table 1. BLT of the Splitter.

Variable	*Equation*	
$h_{1,bc}$	$h_{1,bc} = 100000.0$	binding
$Q_{3,bc}$	$Q_{3,bc} = 2.0$	binding
$Q_{2,bc}$	$Q_{2,bc} = 1.0$	binding
$P_{1,bc}$	$P_{1,bc} = 300000.0$	binding
Q_1	$0 = Q_1 - Q_2 - Q_3$	
Q_2	$Q_2 = Q_{2,bc}$	
Q_3	$Q_3 = Q_{3,bc}$	
P	$P = P_{1,r}$	
P_1	$P_1 = \left(P_{1,l} + P_{1,r}\right)/2$	
P_2	$P_2 = \left(P_{2,l} + P_{2,r}\right)/2$	
P_3	$P_3 = \left(P_{3,l} + P_{3,r}\right)/2$	
$P_{1,l}$	$P_{1,l} = P_{1,bc}$	
$P_{1,r}$	$P_{1,l} - P_{1,r} = k_1.Q_1^2$	
$P_{2,l}$	$P = P_{2,l}$	
$P_{2,r}$	$P_{2,l} - P_{2,r} = k_2.Q_2^2$	
$P_{3,l}$	$P = P_{3,l}$	
$P_{3,r}$	$P_{3,l} - P_{3,r} = k_3.Q_3^2$	
h	$h_2 = h$	

h_1	$h_1 = h_{1,bc}$
h_2	$0 = h_1 \cdot Q_1 - h_2 \cdot Q_2 - h_3 \cdot Q_3$
h_3	$h_3 = h$
T	$h = c_p \cdot T + b \cdot P$
T_1	$h_1 = c_p \cdot T_1 + b \cdot P_1$
T_2	$h_2 = c_p \cdot T_2 + b \cdot P_2$
T_3	$h_3 = c_p \cdot T_3 + b \cdot P_3$

The BLT is obtained by assigning to each variable z of $\mathcal{M}$ the equation E_z that computes it. Because there is a bijection between the set of equations and the set of variables in $\mathcal{M}$, each equation E_z can be uniquely labelled by the name of the variable z that it solves. Therefore, the term Equation z will refer to the equation that solves z, as established by the BLT. The BLT of the Splitter is given in Table 1. To avoid any confusion between variables and equations, Equation z will be denoted $\breve{z}$. For instance, $\breve{Q}_1$ stands for equation $0 = Q_1 - Q_2 - Q_3$, cf. Table 1. A binding is a fixed value assigned to a variable (it is not an equation although it can be found in the BLT).

In the following, the BLT of model $\mathcal{M}$ will also be denoted by $\mathcal{M}$ because the BLT of $\mathcal{M}$ contains all the equations of $\mathcal{M}$. The set $\mathcal{M}'$ is the set $\mathcal{M}$ without the binding equations. In the example:

$$\mathcal{M}' = \mathcal{M} - \{h_{1,bc}, Q_{3,bc,} Q_{2,bc,} P_{1,bc}\}$$

The overall principle of the extraction algorithm is shown in Figure 2. The algorithm starts from set $\mathcal{C}'$ which is the set of equations of $\mathcal{M}'$ that compute the variables of interest:

$$\mathcal{C}' = \breve{x} \cap \mathcal{M}' \tag{39}$$

Let us assume that the variables of interest are:

$$x = \{x_i\} = \{Q_1, Q_2, Q_3, P_1, P_2, P_3, T_1, T_2, T_3, T\}.$$

Then:

$$\mathcal{C}' = \{\breve{x}_i\} \cap \mathcal{M}' = \{\breve{Q}_1, \breve{Q}_2, \breve{Q}_3, \breve{P}_1, \breve{P}_2, \breve{P}_3, \breve{T}_1, \breve{T}_2, \breve{T}_3, \breve{T}\}.$$

For each equation $\breve{x}_i$ in set $\mathcal{C}'$, set $\mathcal{S}_i$ is built by finding the equations in $\mathcal{M}'$ that compute the intermediate variables y_{ij} involved in $\breve{x}_i$ as a function of the variables of interest which are known variables. This procedure is called the chain rule in the following. Its formal specification is shown in Listing 1.

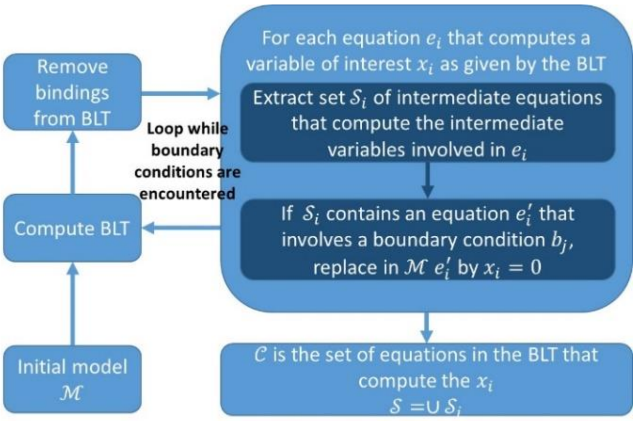

For each equation e_i that computes a variable of interest x_i as given by the BLT
Remove bindings from BLT
Extract set $\mathcal{S}_i$ of intermediate equations that compute the intermediate variables involved in e_i
Loop while boundary conditions are encountered
Compute BLT
If $\mathcal{S}_i$ contains an equation e_i' that involves a boundary condition b_j, replace in $\mathcal{M}$ e_i' by $x_i = 0$
Initial model $\mathcal{M}$
$\mathcal{C}$ is the set of equations in the BLT that compute the x_i
$\mathcal{S} = \cup \mathcal{S}_i$

Figure 2. Principle of the extraction algorithm

Listing 1. Procedure for extracting set $\mathcal{S}_i$

```
set S = empty set;
set V_eq = set of intermediate variables y in equation $\breve{x}_i$;
call extract (S, V_eq, status, fail_eq);
// S contains $\mathcal{S}_i$, status contains $r(\breve{x}_i)$, fail_eq contains $\widetilde{b(\breve{x}_\imath)}$
 define procedure extract (S, V, status, fail_eq)
    set status = SUCCEED;
    for each variable y in V
       if v is a boundary condition then
          set status = FAIL;
       else
          eq = equation in BLT – {bindings}  that computes y;
          if eq exists and eq is not in S then
             insert eq into S;
             set V_eq = set of intermediate variables y in eq;
             call extract (S, V_eq, status, fail_eq);
             if status == FAIL then
                set fail_eq = eq;
                exit procedure;
             end if;
          end if;
       end if;
    end for;
end procedure;
```

Let us apply the chain rule to equation $\breve{T}_3$. Equation $\breve{T}_3$ involves intermediate variables P_3 and h_3. This is denoted by $\breve{T}_3 \rightarrow \breve{P}_3$ and $\breve{T}_3 \rightarrow \breve{h}_3$. Then carrying on with this chain rule yields:

$$\breve{T}_3 \rightarrow \breve{h}_3 \rightarrow \breve{h}_2 \rightarrow \breve{h}_1 \rightarrow \breve{h}_{1,bc} \rightarrow stop.$$

The chain rule is stopped because $h_{1,bc}$ is a boundary condition that cannot be included in the data reconciliation problem. The outcome of the chain rule applied to equation $\breve{x}_i$ is denoted $r(\breve{x}_i)$. The boundary condition that made the chain rule fail for equation $\breve{x}_i$ is denoted $b(\breve{x}_i)$. If the outcome is positive, i.e. if no boundary condition has been encountered, then $r(\breve{x}_i) = true$. Else $r(\breve{x}_i) = false$. Then:

$$\mathcal{S}_i = \{\breve{y}_{ij} | r(\breve{x}_i) = true\} \tag{40}$$

Then $\mathcal{S}_{\breve{T}_3} = \emptyset$.

When applying the chain rule for all equations in set $\mathcal{C}'$, it turns out that $\breve{Q}_1$ is the only equation for which the chain rule is not stopped, in this case because $\breve{Q}_1$ does not involve any intermediate variable. As $\breve{Q}_1$ does not involve any intermediate variable, $\mathcal{S}_{\breve{Q}_1} = \emptyset$.

Set $\mathcal{S}$ is the union of all sets $\mathcal{S}_i$:

$$\mathcal{S} = \cup \mathcal{S}_i \tag{41}$$

Then $\mathcal{S} = \emptyset$.

Set $\mathcal{C}$ contains all equations $\breve{x}_i$ of $\mathcal{C}'$ whose associated set $\mathcal{S}_i$ has been completed without stopping the chain rule. $\breve{Q}_1$ is the only equation that complies with this rule, thus

$$\mathcal{C} = \{\breve{Q}_1\}$$

which corresponds to the mass balance equation.

The variables of interest that can be reconciled are the variables of interest involved in set $\mathcal{C}$ or set $\mathcal{S}$:

$$\bar{\bar{x}} = \{x_i \in x | x_i \in \mathcal{C} \cup \mathcal{S}\} \tag{42}$$

where $x_i \in \mathcal{C} \cup \mathcal{S}$ means that x_i is involved in at least one equation of $\mathcal{C} \cup \mathcal{S}$. Then $\bar{\bar{x}} = \{Q_1, Q_2, Q_3\}$. This means that all variables of interest in $x - \bar{\bar{x}}$ cannot be reconciled:

$$x - \bar{\bar{x}} = \{P_1, P_2, P_3, T_1, T_2, T_3, T\}$$

A larger set $\bar{\bar{x}}$, and thus a higher redundancy level, should be achievable as the model contains momentum and energy balance equations which are appropriate to reconcile pressures and temperatures. Therefore, the algorithm should carry on in order to get into $\bar{\bar{x}}$ as many variables of interest as possible.

To go forward, the idea is to replace in model $\mathcal{M}$ the boundary condition $b(\breve{x}_i)$ that made the chain rule fail for variable of interest x_i by the variable of interest x_i itself. The exact rule is to replace the equation $\overparen{b(\breve{x}_i)}$ that computes $b(\breve{x}_i)$ by equation $(x_i = 0)$. Applying this rule to $\breve{T}_3$ amounts to replacing $\breve{h}_{1,bc}$ by equation $(T_3 = 0)$. Applying this rule to all variables x_i such that $r(\breve{x}_i) = false$ yields a new model $\mathcal{M}_1$:

$$\begin{aligned}\mathcal{M}_1 = \mathcal{M} - \left\{\overparen{b(\breve{x}_i)} \middle| r(\breve{x}_i) = false\right\} \\ + \{(x_i = 0) | r(\breve{x}_i) = false\}\end{aligned} \quad (43)$$

For the example

$$\begin{aligned}\mathcal{M}_1 = \mathcal{M} - \{h_{1,bc}, Q_{3,bc,}Q_{2,bc,}P_{1,bc}\} \\ +\{(T_3 = 0), (P_3 = 0), (Q_3 = 0), (Q_2 = 0)\}\end{aligned}$$

The BLT for $\mathcal{M}_1$ is given in Table 2.

Table 2. BLT of $\mathcal{M}_1$.

Variable	*Equation*	
$h_{1,bc}$	$h_{1,bc} = 100000.0$	binding
$Q_{3,bc}$	$Q_{3,bc} = 2.0$	binding
$Q_{2,bc}$	$Q_{2,bc} = 1.0$	binding
$P_{1,bc}$	$P_{1,bc} = 300000.0$	binding
Q_1	$0 = Q_1 - Q_2 - Q_3$	
Q_2	$Q_2 = 0$	binding
Q_3	$Q_3 = 0$	binding
P	$P = P_{1,r}$	
P_1	$P_1 = (P_{1,l} + P_{1,r})/2$	
P_2	$P_2 = (P_{2,l} + P_{2,r})/2$	
P_3	$P_3 = (P_{3,l} + P_{3,r})/2$	
$P_{1,l}$	$P_{1,l} = P_{1,bc}$	
$P_{1,r}$	$P_{1,l} - P_{1,r} = k_1.Q_1^2$	
$P_{2,l}$	$P = P_{2,l}$	
$P_{2,r}$	$P_{2,l} - P_{2,r} = k_2.Q_2^2$	
$P_{3,l}$	$P = P_{3,l}$	
$P_{3,r}$	$P_{3,l} - P_{3,r} = k_3.Q_3^2$	
h	$h = c_p \cdot T + b \cdot P$	
h_1	$0 = h_1 \cdot Q_1 - h_2 \cdot Q_2 - h_3 \cdot Q_3 + W$	
h_2	$h_2 = h$	
h_3	$h_3 = h$	
T	$T = 0$	binding
T_1	$h_1 = c_p \cdot T_1 + b \cdot P_1$	
T_2	$h_2 = c_p \cdot T_2 + b \cdot P_2$	
T_3	$T_3 = 0$	binding

We now reapply the extraction algorithm to $\mathcal{M}_1$.

$$\mathcal{M}_1' = \mathcal{M}_1 - \{h_{1,bc}, Q_{3,bc,}Q_{2,bc,}P_{1,bc}, Q_2, Q_3, T, T_3\}$$

$$\mathcal{C}_1' = \{\breve{x}_i\} \cap \mathcal{M}_1' = \{\breve{Q}_1, \breve{P}_1, \breve{P}_2, \breve{P}_3, \breve{T}_1, \breve{T}_2\}.$$

Notice now that $\mathcal{C}_1'$ is smaller than $\mathcal{C}'$ by 4 units. Extracting set $\mathcal{S}$ yields:

$$\breve{Q}_1 \rightarrow success$$

$$\breve{P}_1 \rightarrow \breve{P}_{1,l} \rightarrow \breve{P}_{1,r} \rightarrow \breve{P} \rightarrow \breve{P}_{3,l} \rightarrow \breve{P}_{3,r} \rightarrow success$$

$$\breve{P}_2 \rightarrow \breve{P}_{2,l} \rightarrow \breve{P} \rightarrow \breve{P}_{3,l} \rightarrow \breve{P}_{3,r} \rightarrow \breve{P}_{2,r} \rightarrow success$$

$$\breve{P}_3 \rightarrow \breve{h}_3 \rightarrow \breve{h} \rightarrow \breve{P} \rightarrow \breve{P}_{3,l} \rightarrow \breve{P}_{3,r} \rightarrow success$$

$$\breve{T}_1 \rightarrow \breve{h}_1 \rightarrow \breve{h}_3 \rightarrow \breve{h} \rightarrow \breve{P} \rightarrow \breve{P}_{3,l} \rightarrow \breve{P}_{3,r} \rightarrow \breve{h}_2 \rightarrow success$$

$$\breve{T}_2 \rightarrow \breve{h}_2 \rightarrow \breve{h} \rightarrow \breve{P} \rightarrow \breve{P}_{3,l} \rightarrow \breve{P}_{3,r} \rightarrow success$$

Then:

$$\mathcal{C} = \{\breve{Q}_1, \breve{P}_1, \breve{P}_2, \breve{P}_3, \breve{T}_1, \breve{T}_2\}$$

and:

$$\mathcal{S}_{\breve{Q}_1} = \emptyset$$

$$\mathcal{S}_{\breve{P}_1} = \{\breve{P}_{1,l}, \breve{P}_{1,r}, \breve{P}, \breve{P}_{3,l}, \breve{P}_{3,r}\}$$

$$\mathcal{S}_{\breve{P}_2} = \{\breve{P}_{2,l}, \breve{P}, \breve{P}_{3,l}, \breve{P}_{3,r}, \breve{P}_{2,r}\}$$

$$\mathcal{S}_{\breve{P}_3} = \{\breve{h}_3, \breve{h}, \breve{P}, \breve{P}_{3,l}, \breve{P}_{3,r}\}$$

$$\mathcal{S}_{\breve{T}_1} = \{\breve{h}_1, \breve{h}_3, \breve{h}, \breve{P}, \breve{P}_{3,l}, \breve{P}_{3,r}, \breve{h}_2\}$$

$$\mathcal{S}_{\breve{T}_2} = \{\breve{h}_2, \breve{h}, \breve{P}, \breve{P}_{3,l}, \breve{P}_{3,r}\}$$

$$\begin{aligned}\mathcal{S} &= \mathcal{S}_{\breve{Q}_1} \cup \mathcal{S}_{\breve{P}_1} \cup \mathcal{S}_{\breve{P}_2} \cup \mathcal{S}_{\breve{P}_3} \cup \mathcal{S}_{\breve{T}_1} \cup \mathcal{S}_{\breve{T}_2} \\ &= \{\breve{P}_{1,l}, \breve{P}_{1,r}, \breve{P}, \breve{P}_{3,l}, \breve{P}_{3,r}, \breve{P}_{2,l}, \breve{P}_{2,r}, \breve{h}_3, \breve{h}, \breve{h}_1, \breve{h}_2\}\end{aligned}$$

Notice that the original model has a 21-equation algebraic system to be solved, whereas the extracted system for data reconciliation has only an 11-equation algebraic system to be solved (in fact two separate 11-equation algebraic systems, one for solving set $\mathcal{S}$ and the other for computing the Jacobian matrix of set $\mathcal{C}$, cf. resp. Equations (36) and (38)).

The variables of interest that can be reconciled are those who appear in set $\mathcal{C}$ or in set $\mathcal{S}$. All variables of interest appear in set $\mathcal{C}$ or in set $\mathcal{S}$, therefore all variables of interest can be reconciled:

$$\bar{\bar{x}} = \{\bar{\bar{Q}}_1, \bar{\bar{Q}}_2, \bar{\bar{Q}}_3, \bar{\bar{P}}_1, \bar{\bar{P}}_2, \bar{\bar{P}}_3, \bar{\bar{T}}_1, \bar{\bar{T}}_2, \bar{\bar{T}}_3, \bar{\bar{T}}\}$$

The redundancy level is 6 and the size of the algebraic system is divided by approximately 2. The extraction algorithm is completed■

If the set of variables of interest is

$$x = \{Q_1, Q_2, Q_3\},$$

then:

$$\mathcal{C} = \{\breve{Q}_1\}$$

$$\mathcal{S} = \emptyset$$

$$\bar{\bar{x}} = \{\bar{\bar{Q}}_1, \bar{\bar{Q}}_2, \bar{\bar{Q}}_3\}$$

All variables of interest can be reconciled, the redundancy level is 1 and there is no algebraic system to be solved.

If the set of variables of interest is

$$x = \{P_1, P_2, P_3\},$$

then:

$$\mathcal{C} = \emptyset$$
$$\mathcal{S} = \emptyset$$
$$\bar{\bar{x}} = \emptyset$$

No variable of interest can be reconciled. This is because there are too many boundary conditions related to the variables of interest. The data reconciliation problem is thus ill-posed.

3.7 Numerical Results

The inputs for the numerical computations are the measured values $\hat{x}_i$, the weights or half-width confidence intervals $w_{\hat{x},i}$ and the correlation matrix coefficients $r_{\hat{x},i,j}$.

In the following examples, the correlation matrix coefficients are equal to zero: $r_{\hat{x},i,j} = 0$ for $i \neq j$. Then only the measured values and weights are provided.

In the computation of Table 3 and Table 4, the values for the parameters are $k_1 = k_2 = k_3 = 1\ \mathrm{bar.kg^{-2}.s^{-2}}$, $c_p = 4.2\ \mathrm{kJ.kg^{-1}.°C^{-1}}$, $b = 0.19\ \mathrm{kJ.bar^{-1}}$ and $W = 1\ \mathrm{MW}$. One can verify that the reconciled weights are smaller than the measured weights.

Table 3. Inputs for the Splitter

Variable	*Measured value*	*Weight*
Q_1	2.5 kg/s	0.196
Q_2	1.15 kg/s	0.196
Q_3	1.25 kg/s	0.196
P_1	6.1 bar	0.392
P_2	2.55 bar	0.392
P_3	2.45 bar	0.392
T	114 °C	1.96
T_1	19 °C	1.96
T_2	113 °C	1.91
T_3	115 °C	1.91

Table 4. Reconciled values for the Splitter

Variable	*Reconciled value*	*Reconciled weight*	*Individual test*
Q_1	2.49413 kg/s	0.0521606	*true*
Q_2	1.20022 kg/s	0.120607	*true*
Q_3	1.2939 kg/s	0.120293	*true*
P_1	6.29266 bar	0.250971	*true*
P_2	2.46206 bar	0.274044	*true*
P_3	2.34524 bar	0.276126	*true*
T	114.124 °C	1.08222	*true*
T_1	18.5211 °C	1.7887	*true*
T_2	114.157 °C	1.08159	*true*
T_3	114.162 °C	1.08158	*true*

Listing 2. Tagging the boundary conditions

```
parameter Real Q0=100 "Fluid mass flow rate"
annotation(__OpenModelica_BoundaryCondition =
true);
  parameter Real h0=100000 "Fluid specific
enthalpy"
annotation(__OpenModelica_BoundaryCondition =
true);
```

In the computation of Table 5, only the mass flow rates are reconciled with the measured values of Table 3. One can verify that the reconciled weights are smaller than the measured weights but are larger than the values obtained when reconciling the mass flow rates with the pressures and the temperatures. This is consistent with the fact that more information leads to better accuracy.

Table 5. Reconciled values for the Splitter

Variable	*Reconciled value*	*Reconciled weight*	*Individual test*
Q_1	2.46667 kg/s	0.160033	*true*
Q_2	1.18333 kg/s	0.160033	*true*
Q_3	1.28333 kg/s	0.160033	*true*

In both calculations, the χ^2-test of Condition (22) is satisfied.

3.8 Interface with Modelica in OpenModelica

OpenModelica is an open source tool for the modelling and simulation of Modelica models (Fritzson et al., 2020). The data reconciliation interface with Modelica newly implemented in OpenModelica aims at giving the possibility to perform data reconciliation on a validated Modelica model without having to modify the model.

To perform data reconciliation, three to four actions are necessary:

1. Tag the boundary conditions.
2. Tag the variables of interest.
3. Provide the measured values and weights.
4. If necessary, provide the correlation coefficients.

In Modelica libraries, boundary conditions are most often to be found in specialized components such as mass flow rate, pressure and temperature sources and sinks. The tagging of boundary conditions is therefore permanent

Listing 3. Tagging the variables of interest

```
model TSP_Splitter_DR
  TSP_Splitter splitter(
     pipe1(Q(uncertain = Uncertainty.refine)),
     pipe2(Q(uncertain = Uncertainty.refine)),
     pipe3(Q(uncertain = Uncertainty.refine)),
     pipe1(Pm(uncertain = Uncertainty.refine)),
     pipe2(Pm(uncertain = Uncertainty.refine)),
     pipe3(Pm(uncertain = Uncertainty.refine)),
     volume(T(uncertain = Uncertainty.refine)),
     pipe1(T(uncertain = Uncertainty.refine)),
     pipe2(T(uncertain = Uncertainty.refine)),
     pipe3(T(uncertain = Uncertainty.refine)));
equation
end TSP_Splitter_DR;
```

Listing 4. Csv file for the measured values

```
Variable name; Measured value; Weight
splitter.pipe1.Q; 2.50; 0.196
splitter.pipe2.Q; 1.15; 0.196
splitter.pipe3.Q; 1.25; 0.196
splitter.pipe1.Pm; 6.1e5; 0.392e5
splitter.pipe2.Pm; 2.55e5; 0.392e5
splitter.pipe3.Pm; 2.45e5; 0.392e5
splitter.volume.T; 387; 1.96
splitter.pipe1.T; 292; 1.96
splitter.pipe2.T; 386; 1.91
splitter.pipe3.T; 388; 1.91
```

and should not interfere with the usual simulation activities (i.e. DAE integration from initial conditions). It also should not prevent the components to be used with tools that do not support data reconciliation. Therefore, the tagging of boundary conditions is done with a special annotation. In Listing 2, two boundary conditions are tagged: the fluid mass flow rate and the fluid specific enthalpy.

The variables of interest are tagged by the user in a different way for each data reconciliation problem. The tags should not modify the original Modelica model, and appropriate checks must be done by the tool to verify that the tags refer to existing and eligible variables. Therefore, the tagging of variables of interest is done with a special modifier. In Listing 3, the name of the original model of the splitter is TSP_Splitter. One instance of TSP_Splitter is placed in model TSP_Splitter_DR that is especially created to perform data reconciliation on the model TSP_Splitter. The variables of interest in model TSP_Splitter are tagged from model TSP_Splitter_DR using the `uncertain` modifier. Checks are performed by the tool to ensure that the tagged variables are eligible to be tagged as variables of interest. Thus, the original model is not modified and can be used in any Modelica tool. The embedding model can however only be used in tools supporting data reconciliation.

The measured values and weights are provided in a csv file with 3 columns, cf. Listing 4 where the values are given in SI units.

The correlation matrix is provided in an optional separate csv file. No correlation matrix file is given for the example.

3.9 Data Reconciliation with ThermoSysPro

ThermoSysPro is a Modelica library for the modelling and simulation of power plants and energy systems at large (El Hefni and Bouskela, 2019).

The ThermoSysPro model of the Splitter is shown in Figure 3. It is equivalent to the model proposed in Section 3.3, the equations being dispatched in the following specialized components: `SourceP` (pressure source), `SinkQ` (mass flow rate sink), `SingularPressureLoss` (pipes), `VolumeBTh` (splitter volume), `HeatSource` (thermal power). Therefore, for instance the variable `splitter.volume.T` in Listing 4 is the temperature T inside the splitter volume. The Modelica model has a total of 121 variables and 121 equations.

The model components used in the Splitter model are modified as follows.

1. `SourceP` and `SinkQ`: The boundary conditions are tagged as shown in Listing 2.
2. `VolumeBTh`: The temperature is computed from the specific enthalpy with equation $\breve{T}$ in Table 1.
3. `SingularPressureLoss`: The pressure loss is computed from the mass flow rate with equation $\breve{P}_{i,r}$ in Table 1 for $i = 1, 2, 3$. The temperature is computed from the specific enthalpy with equation $\breve{T}_i$ in Table 1 for $i = 1, 2, 3$. The mass flow reversal equation in each pipe which according to the upwind scheme should be (El Hefni and Bouskela, 2019, Eq. 4.114)
$$h_i = \begin{cases} h_{i,l} & \text{if } Q_i \geq 0 \\ h_{i,r} & \text{if } Q_i < 0 \end{cases}$$
is replaced by
$$h_i = h_{i,l}$$
under the assumption that mass flow rates are positive, in order to avoid dependencies with boundary conditions $h_{2,r}$ and $h_{3,r}$ so that Condition (35) can be satisfied and all temperatures can be reconciled. This replacement was performed manually but could be done automatically as the sign of Q_i is fixed and known beforehand.

Modifications in points 2 and 3 above are made to avoid numerical difficulties when solving the algebraic equations.

The extracted model has 10 variables to be reconciled, 6 auxiliary conditions and 41 intermediate equations. Therefore, the size of the algebraic system to be solved is divided by 3.

The results of the data reconciliation computation with ThermoSysPro are shown in Table 6. They are different from the results obtained without ThermoSysPro displayed in Table 4 because the two splitter models are equivalent, but not identical, however results stay within their confidence intervals.

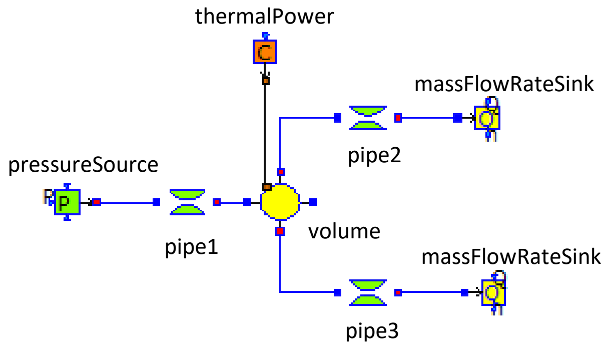


Figure 3. ThermoSysPro model of the splitter

Table 6. Reconciled values with ThermoSysPro

Variable	*Reconciled value*	*Reconciled weight*	*Individual test*
Q_1	2.49327 kg/s	0.0521266	*true*
Q_2	1.19978 kg/s	0.120619	*true*
Q_3	1.29349 kg/s	0.120306	*true*
P_1	6.29090 bar	0.250917	*true*
P_2	2.46296 bar	0.274003	*true*
P_3	2.34614 bar	0.276086	*true*
T	113.987 °C	1.08228	*true*
T_1	18.351 °C	1.78902	*true*
T_2	114.019 °C	1.08165	*true*
T_3	114.025 °C	1.08163	*true*

4 Conclusion and Future Work

Performing data reconciliation on a Modelica model requires to extract the auxiliary conditions that constrain the variables of interest from the Modelica model and remove the boundary conditions which are related to the variables of interest. This is why, contrary to data assimilation (Corona Mesa-Moles et al., 2019), data reconciliation cannot be performed on a Modelica model in a black box manner.

An extraction algorithm has been presented and applied on a small example with 121 equations built with the ThermoSysPro library. The extracted model has 10 variables to be reconciled, 6 auxiliary conditions and 41 intermediate equations, which shows that the extraction algorithm reduces significantly the size of the algebraic system to be solved (here by a factor of 3). Some modifications were required in the ThermoSysPro library to perform the numerical computations. They mainly consisted in simplifying the equations that compute the fluid properties and those that compute the specific enthalpy according to the upwind scheme. These simplifications around the measured values are acceptable because data reconciliation assumes that the differences between the measured and reconciled values are small.

This work demonstrates that Modelica can be used for other purposes than for initial value problems, and that the knowledge embedded in existing models can be utilized also for data processing.

Future work will consist in verifying that the method presented in the paper is applicable to larger models. A preliminary experiment was conducted on the model of a secondary side of a nuclear power plant to compute the reactor nominal power. The model has 2002 variables and equations and 26 variables to be reconciled. The extraction algorithm produced 23 auxiliary conditions and 553 intermediate equations. This shows that a higher level of redundancy and algebraic system reduction can be achieved on larger models, making paradoxically data reconciliation more efficient to perform on larger Modelica models than on smaller ones.

Future work will also consist in computing the values of the boundary conditions from the reconciled values, have the new method certified by the VDI 2048 standard committee and integrate the new modifier for tagging the variables of interest into the Modelica standard.

Acknowledgements

This work was partially supported by DGE (France) and Vinnova (Sweden) in the scope of the ITEA 3 OPENCPS project.

References

Bai, S. and Thibault, J. (2010). "*Dynamic Data Reconciliation*". VDM Verlag. ISBN: 978-3-639-21779-7.

Bedjaoui, N., Litrico, X., Koenig, D., Ribot-Bruno, J. and Malaterre, P.O. (2008). "*Static and Dynamic Data Reconciliation for an Irrigation Canal*". Journal of Irrigation and Drainage Engineering, 134: 778-787.

Belsim (2021). "*VALI Software Suite*". URL: https://belsim.com/vali-software/ (visited on 2021-04-14).

Corona Mesa-Moles, L., Argaud, J.P., Jardin, A., Benssy, A. and Dong, Y. (2019). "*Robust Calibration of Complex ThermosysPro Models using Data Assimilation Techniques: Application on the Secondary System of a Pressurized Water Reactor*". Linköping Electronic Conference Proceedings 157:56, s. 8.

Dutfoy A., Dutka-Malen I., Lebrun R. et al. (2009). "*OpenTURNS, an Open Source Initiative to Treat Uncertainties, Risks'N Statistics in a Structured Industrial Approach*". In: *41èmes Journées de Statistique, SFdS, Bordeaux*.

El Hefni, B. and Bouskela, D. (2019). "*Modeling and Simulation of Thermal Power Plants with ThermoSysPro*". Springer. ISBN: 978-3-030-05104-4.

Fritzson, P., Pop, A., Abdelhak, K., Asghar, A. et al. (2020). "*The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development*". Modeling, Identification and Control. 2020;41(4):241-295.

Langenstein, M., Jansky, J., Laipple, B., Grauf, E., Schak, H., & Eitschberger, H. (2004). "*Finding megawatts in nuclear power plants with data reconciliation*". In: *Proceedings of the 12th International Conference on Nuclear Engineering*. Vol. 2. American Society of Mechanical Engineers – ASME.

Modelica Association (2021). "*Modelica – A Unified ObjectOriented Language for Systems Modeling. Language Specification Version 3.5*". Tech. rep. Linköping: Modelica Association. URL: https://www.modelica.org/documents/MLS.pdf.

VDI - Verein Deutscher Ingenieure (2017). VDI 2048 – Blatt 1 "*Control and quality improvement of process data and their uncertainties by means of correction calculation for operation and acceptance tests, VDI-Handbuch Energietechnik*", ICS: 17.020, 27.010.

DOI
10.3384/ecp21181453

Use of Modelica to predict risk of Covid-19 infection in indoor environments

Arnav Pathak Kilian Schneider Victor Norrefeldt

Fraunhofer-Institute for Building Physics, Fraunhoferstr. 10, D-83626 Valley, Germany
victor.norrefeldt@ibp.fraunhofer.de

Abstract

In the light of the Sars-CoV-2 pandemic, the dispersion process of respiratory droplets released by potentially infected persons has been investigated in many studies using highly reliable but time consuming CFD methods. With such simulations social distancing, wearing masks and shifts in ventilation systems could be justified. This work focuses on the same topic but uses the validated Velocity Propagating Zonal Model (VEPZO) instead of CFD simulations. It is implemented in Modelica and allows fast simulation of the indoor environment on a coarse grid which in many cases is a superior alternative to complex CFD simulations in the trade-off between effort and detail of the result. Based on the temperature and airflow distribution, this model can be used to predict the dispersion of aerosols in enclosed spaces and thus the relative risk of Covid-19 infection. For model verification, a documented outbreak in a restaurant in Guangzhou is being investigated. An improved ventilation pattern to contain viral load more locally is developed.

1 . Introduction

Covid-19 is mainly transmitted from human-to-human through inhalation of respiratory droplets. Transmission of respiratory infectious diseases has been studied in different disciplines for decades. Computational Fluid Dynamics (CFD) simulation is an accurate and reliable method to predict airflow patterns and thus lately the risk of airborne cross infection in a crowded indoor environment with an infected index patient. Zhang and Li (2012) studied the dispersion process of respiratory droplets released by a coughing person in a high speed rail cabin using CFD simulations. Four cases of different air supply and exhaust locations are reviewed. The droplets' dispersion characteristics and the maximum dispersion distances under specified ventilation conditions are investigated. This study demonstrates the potential of improving cabin air ventilation for infection control. In Goscé et al. (2014) the authors show the dependency of walking speed on the rate of infection in crowded underground corridors. Recent discussions investigated infection risks also in aircraft cabins and other indoor environments such as offices, classrooms and restaurants. Yan et al. (2017) simulated a whole fully occupied 7 row aircraft cabin and investigated the influence of different ventilation systems on the droplet dispersion for different respiratory activities of a single person. The key particle transport information such as the particle residence time yielded from the Lagrangian tracking process was extracted and integrated into the Wells-Riley equation (Sze To and Chao 2010) in conjunction with CFD predictions. Villafruela et al. (2016) focus on the validation of a 3-D transient CFD model used to predict personal exposure to airborne pathogens and infection risk in a displacement ventilated room. After consideration of different interactions of two exhaling persons, they conclude that numerical simulations have the capacity to analyse the dispersion of exhaled contaminants over time.

Melikov (2020) postulated a paradigm shift in ventilation design which is needed in order to respond to the current need. State of the art ventilation is mainly based on mixing air distribution. This does neither provide efficient removal of polluted air exhaled by occupants before mixing nor the direct supply of clean air to the breathing zone.

The investigated case of this paper is a so called super spreader event taken place in a restaurant in Guangzhou, China (Lu et al. 2020). With the help of the Velocity Propagating Zonal model (VEPZO) (Norrefeldt et al. 2012) the case could be reconstructed and the infection spread traced back to a poorly ventilated dining room. Furthermore an alternative to the existing ventilation system is developed in order to contain the spread of potentially infectious aerosols.

2 . Methods

2.1 Implementation of the VEPZO model

Zonal models allow simpler and faster estimation of indoor climate than complex CFD simulations. They use similar mathematical theory, but subdivide the space more coarsely into 10 to 100 zones exchanging air through flow paths. These are usually based on the Bernoulli-equation, causing numerical issues and a lack of precision at a zero pressure difference due to the square root function. The Velocity Propagating Zonal Model (VEPZO) is an advanced type of zonal model, developed at the Fraunhofer-Institute for Building Physics. By introducing airflow velocity vectors and making them a property of the zone, the airflow velocity

DOI
10.3384/ecp21181463

is no longer dissipated once it has entered a zone but propagates into space. Therefore the zonal formulation is still valid in areas with driving flows due to jets or plumes and overcomes the need for correlation formulation in those areas.

The two main components of the VEPZO model are a zone model and a flow model (Figure 1). The zone (cube) and the flow (grey rectangle) models are connected by ports (rhombs) to form a room. These ports allow the exchange of relevant information between the flow and the zone model. The flow models have two ports to connect adjacent zones. Each zone has six ports, one for each boundary. A Boolean parameter is assigned to each port to make the distinction whether the port is connected to a flow model or whether there is no flow because the zone is adjacent to a room boundary surface.

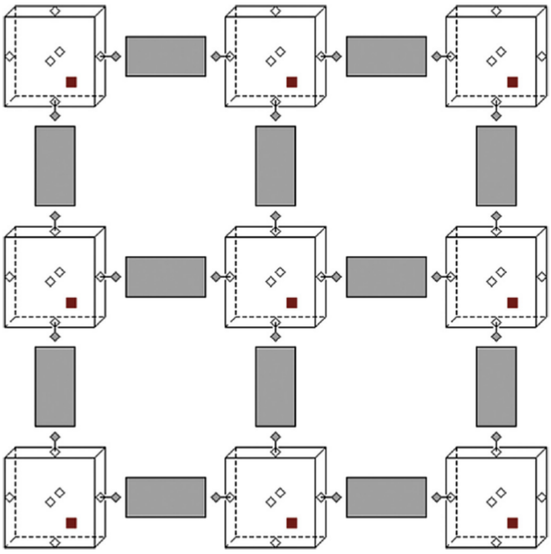
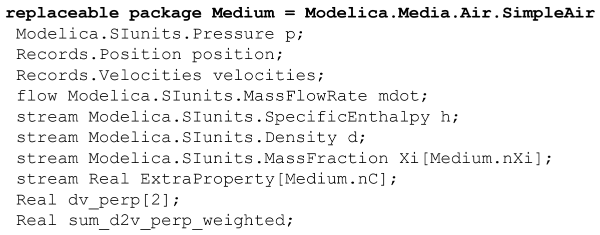


Figure 1: Zonal model in x-z direction (y not shown); cubes: zones; grey rectangles: flows; rhombs: airflow ports; red solid squares: heat ports.

Each zone has a heat port (red square) allowing exchanges with models of other components like e.g. heat sources or walls to assess the local impact of these heat releases on the airflow pattern and the temperature distribution in a room. Models computing the interaction of air properties like density, pressure, temperature or specific enthalpy are available in the Modelica.Media library and are used in the VEPZO model (Figure 2). Depending on the application, the air model can be changed from dry to moist air. Pollutants can be taken into account by adding a tracer substance (ExtraProperty) to the Media model.



```
replaceable package Medium = Modelica.Media.Air.SimpleAir
 Modelica.SIunits.Pressure p;
 Records.Position position;
 Records.Velocities velocities;
 flow Modelica.SIunits.MassFlowRate mdot;
 stream Modelica.SIunits.SpecificEnthalpy h;
 stream Modelica.SIunits.Density d;
 stream Modelica.SIunits.MassFraction Xi[Medium.nXi];
 stream Real ExtraProperty[Medium.nC];
 Real dv_perp[2];
 Real sum_d2v_perp_weighted;
```

Figure 2: Implementation of connector, position and velocities

2.2 Zonal Model

The main task of the zone model is to compute the mass and enthalpy balance and air properties (density, enthalpy, pressure, temperature, etc.) using air models of Modelica.Media. Furthermore it determines a characteristic velocity and viscous losses.

The zone model contains interfaces to various types of fluxes, including those of air, gases and particles.

Air contained in a zone is assumed to be perfectly mixed. The mass conservation takes into account the amount of air exchanged with adjacent zones and airflows provided by various sources or sinks (ventilation, openings, etc.) in the zone. Heat flows due to convection to walls or heat sources contained in the zone are added to the thermal energy balance. A new feature of the VEPZO model is that a characteristic velocity vector (u,v,w) is assigned to the zones. Knowing the mass flow and its direction across each of the zone's surfaces, the flow velocity across these surfaces is determined. The zone shares the information about its characteristic velocity with the flow models surrounding it. This enables the VEPZO model to propagate the airflow velocity throughout the room without needing special correlations like jets or plumes.

2.3 Flow model

The main task of the flow model is to compute the airflow rate between two adjacent zones. Furthermore, the flow models are used to calculate the velocity gradient needed for the calculation of viscous losses. Two adjacent zones are connected by a flow model computing the exchange of air between them. The VEPZO model uses flow models in x-, y- and z directions. The assumption of the VEPZO model is that air only flows along these specific directions. A new feature of the flow model used in the VEPZO model is that the length of a flow path is taken into account. The flow model computes the airflow acceleration or deceleration from the forces acting on it (Figure 3).

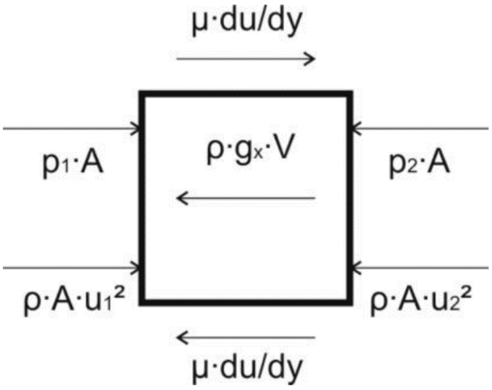


Figure 3: Forces acting on airflow

F_P from pressure differences

Air contained in each zone has a certain pressure. When two zones of common surface A are connected by a flow model they process their pressure information p_1 and p_2. The flow model calculates the resulting force.

$$F_P = -A \cdot \left(p_j - p_i\right) \tag{1}$$

F_M from momentum difference

The characteristic velocity vectors of adjacent zones are processed from the zone model to the flow model. According to the flow direction (x, y or z) the flow model chooses the proper component of the velocity vectors (u_1, v_1, w_1 and u_2, v_2, w_2) to compute the force resulting from the momentum difference between the adjacent zones taking into account the surface A and mean density ρ of the airflow.

Direction	Momentum forces	
x	$F_{M,x} = -\rho \cdot A \cdot \left(u_j^2 - u_i^2\right)$	
y	$F_{M,y} = -\rho \cdot A \cdot \left(v_j^2 - v_i^2\right)$	(2)
z	$F_{M,z} = -\rho \cdot A \cdot \left(w_j^2 - w_i^2\right)$	

Gravitational forces F_G

Gravitational forces occur in the downward direction. To compute the gravitational force, the surface A and the length Δz of the flow path are considered to calculate the volume V of the airflow. Multiplied with the mean density ρ and the acceleration g it yields a force.

$$F_G = -\rho \cdot g \cdot A \cdot \Delta z_{ij} \tag{3}$$

Viscous forces $F_{V,Flow}$

In the selected approach of the VEPZO model, flows are connected and exchange information with zones only. However, to calculate the shear stress, an information exchange between parallel flow models would be necessary. To avoid connections between the flow models, viscous losses are calculated in the zone models but used in the flow models. The characteristic velocity vector provided by zones enables the flow model to calculate the gradient of the two velocity components perpendicular to the flow model direction. For example, a flow model in z-direction can deliver the variation of the characteristic velocities u_1, u_2 and v_1, v_2 in the x and y-directions. If a wall is adjacent to the zone, the velocity at the wall is assumed to be zero. Therefore, the gradient is equal to the characteristic velocity divided by half the distance of the zone's centre from the wall.

The gradient information is transmitted from the flow model to the zone model. In the zone model this gradient causes shear stresses on its boundaries. This shear stress takes into account the dynamic viscosity μ and the derivation of the velocity w.r.t. the two other Cartesian directions. Summing these shear stresses along the boundaries and multiplying them with the surface area yields the viscous forces in the zones.

Direction	Gradients in flow model		Gradients at walls		
x	$\frac{\Delta v_{ij}}{\Delta x_{ij}}$	$\frac{\Delta w_{ij}}{\Delta x_{ij}}$	$\frac{2 \cdot v_i}{\Delta x_i}$	$\frac{2 \cdot w_i}{\Delta x_i}$	
y	$\frac{\Delta u_{ij}}{\Delta y_{ij}}$	$\frac{\Delta w_{ij}}{\Delta y_{ij}}$	$\frac{2 \cdot u_i}{\Delta y_i}$	$\frac{2 \cdot w_i}{\Delta y_i}$	(4)
z	$\frac{\Delta u_{ij}}{\Delta z_{ij}}$	$\frac{\Delta v_{ij}}{\Delta z_{ij}}$	$\frac{2 \cdot u_i}{\Delta z_i}$	$\frac{2 \cdot v_i}{\Delta z_i}$	

The forces acting on a flow path are summed up. This yields the acceleration of the portion of air contained in the flow path connecting two zones:

$$A \cdot \Delta x_{12} \cdot \rho \cdot \dot{u} = F_p + F_{M,x} + F_{Vx,Flow} + F_{G,x} \tag{5}$$

The mass flow is obtained straight forward from the velocity in a flow path. This mass flow information is transmitted to the zone model.

$$\dot{m}_x = \rho \cdot A \cdot u \tag{6}$$

3 . Experimental Validation

The VEPZO model was originally developed to transiently calculate the indoor climate of an aircraft cabin in a short time and to be able to develop an optimal climate for passenger comfort and temperature distribution. For such applications, the simulation was validated in (Norrefeldt et al. 2015). The experimental results used to validate the Thermal Model tests are performed in the Thermal Test Bench that has been set up in the CleanSky (Clean Sky JU 2011) project. Simulation of heated equipment were compared to measurements under various conditions which show that the model predicts equipment temperatures within an accuracy of 1 Kelvin.

For the model setup, the Modelica Thermal Model Generation Tool was implemented. Developed by Pathak et al. (2014) it aims to enable the user to set up a geometrically correct thermal model for complex geometries that allows predicting the impact of heated devices and their location on indoor climate.

To optimize the indoor climate, the VEPZO model has also been coupled with a genetic optimization algorithm. This algorithm optimizes a set of different solutions by combination and selection based on Darwin's theory of evolution. Using the example of a hybrid ventilated classroom (window and additional mechanical ventilation) under cold ambient conditions, an optimal use of tilt windows and the arrangement of heating devices to create optimal comfort and air quality in the occupied area was investigated (Norrefeldt et al. 2013; Reim et al. 2015).

To further validate the VEPZO model, Lindner et al. (2019) evaluated various climatic scenarios in an aircraft galley near an exterior door (Figure 4). The door represents a thermal bridge due to the structural reinforcement of the metallic frame. As a result, the crew often suffers from uncomfortably cold feet. With the help of the VEPZO model, different approaches to locally improve this area were evaluated by simulation and the most promising solutions were implemented in the test setup. Due to the short simulation time of about 10 minutes on a laptop with 2.7 GHz, many different scenarios could be evaluated in a flexible and time-saving way.

Figure 4 shows the comparison between the measured (top) and simulated (bottom) temperature map. The black dots describe sensor locations for the measurement and zone or surface centres in the simulation. The zonal grid is thus more refined than the measurement grid. As the colour maps are interpolated between the points, this geometrical difference leads to different apparent colour gradients.

Comparing the plots show that the temperatures in the left and right bottom corner are accurately predicted. Because the adjacent zone is geometrically closer than the adjacent sensors, the colour interpolation of the plot seemingly shows a larger cold zone in the measurement. In the middle of the galley, a hot air exhaust is implemented at ground level. This leads to increased surface temperature and to an increased temperature of the bottom zone, especially between positions C and D. A similar effect is measured, too. The plots in this area seemingly differ because the zone above is closer than the next sensor in the vertical direction. This leads to a seemingly narrower heated zone in the measured plot than in the simulated one.

At the jumpseat in position D, a warm mounting plume is both simulated and confirmed by measurement.

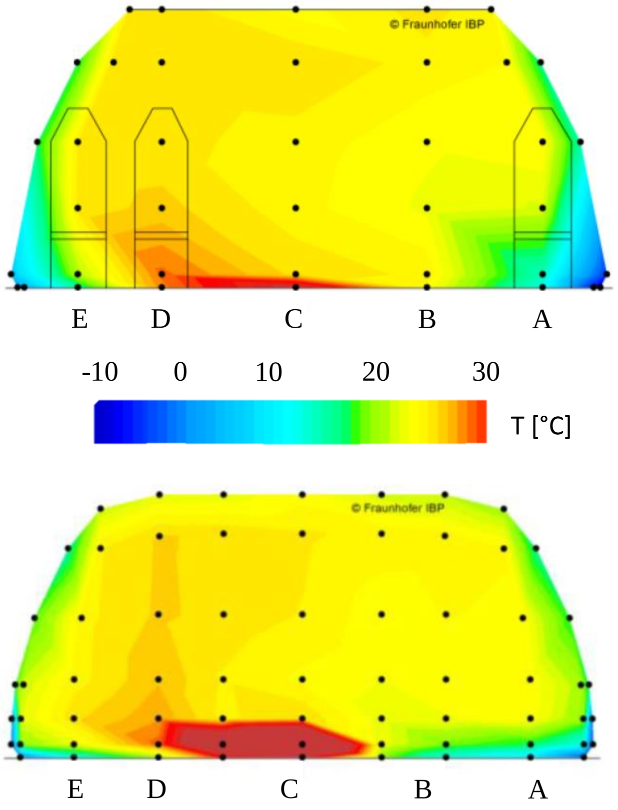


Figure 4: Experimental results of temperature distribution from derived improvement by means of an air heater (top) and results from simulation (bottom)

4 . Extension for the evaluation of the risk of infection with Covid-19

As part of the Fraunhofer vs. Corona initiative, the VEPZO model is extended to include the spread of potentially infectious aerosols and viruses in the indoor environment. The aim is to assess potential Covid-19 infection risks in enclosed spaces. For this purpose, humans become a heat and virus source in the simulation. The viral concentration was added as an extra property to the air medium model. This results in the local distribution of the viral load emitted by an infected person in the room. The source strength is described using the concept of Quanta (Buonanno et al, 2020, Jimenez 2020) and can be adapted to the activity of the person (e.g. 2.3 quanta/h for breathing, 11.4 quanta/h for speaking and 65.1 quanta/h for loud speaking) and to personal protective equipment (none, different types of masks, etc.). The assumption is, that a higher exposure to a predicted concentration of quanta correlates with an increased infection risk.

Modelling can be used to evaluate measures to reduce the viral load. Such devices are functionally described in the model, for example, increased ventilation diluting the quanta concentration or air filtration removing quanta from the aspired air. Product specific performance data, which are determined in the laboratories of Fraunhofer IBP, can also be replicated in these models. Due to the current situation in the Corona pandemic, such a predictive model is a valuable tool and of great importance to be able to prevent possible transmission of infections.

A second concept to tackle the indoor hygiene in the simulation is the concept of age of air. For this, an equally distributed source of an arbitrary tracer gas is distributed in the indoor space and the relative concentrations are compared. In the zonal model, this is achieved by adding an internal source of an “extra property” to each zone. The intensity is proportional to the zone’s volume. In the simulation, areas close to the air inlet will thus have a relatively lower concentration of such tracer and thus a low age of air, whereas poorly ventilated spots or zones further downstream will compute a higher concentration and thus a high age of air.

The difference between both concepts is that the viral emission considers a local source and its spread in space whereas the age of air globally identifies where relatively new respectively old air is available.

5 . Application

A well-documented superspreading event in a restaurant in Guangzhou (Lu et al. 2020) was used as an application and verification example of the modelling. Starting from an index patient, the distribution of the infectious aerosols is predicted. Figure 5 shows the considered case of the restaurant and the ventilation system. In retrospect, nine infections with Covid-19 of members of three families could be attributed to this event (Lu et al. 2020). All infected guest were seated on the left three tables. Said families were in the restaurant at the same time on January 24, 2020, together with 81 other persons (73 guests, 8 employees). The airflow rate was 1 litre per second and per person, divided among a total of five AC units (air conditioning). The air was only circulated, there was no fresh air supply.

The simulation of a two hour exposure with the zonal model with quanta emission took 1 min on a normal notebook (1.6 GHz procrressor with 8GB RAM). The simulation was performed using the Esdirk 23 – order 3 stiff solver with a tolerance of 0.0001 and a fixed time step of 30s. Flags were used for sparse, parallel computation on 4 cores.

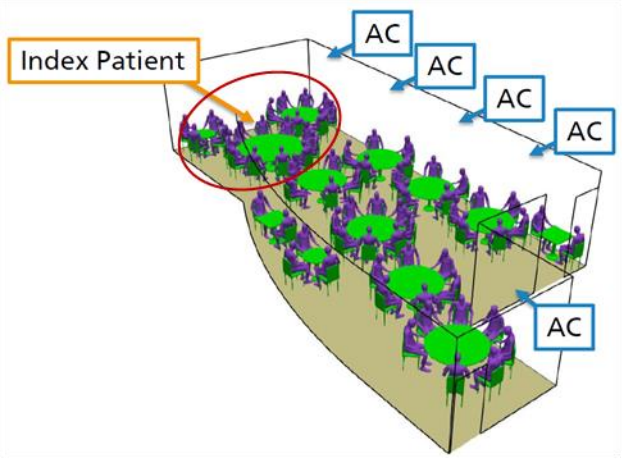


Figure 5: Schematic view of the ventilation system of the considered case and area where infetions occurred (marked in red)

Figure 6 shows the predicted age of air and the area of increased quanta load calculated using the VEPZO model. A challenge for the validation of such models is the absence of measurement data for the distribution of viral load. However, the comparison of the predicted area of increased quanta load with the documented infections shows that both are well inline.
The poor ventilation and low air exchange rate in this case played a central role, which prevented rapid renewal of the room air from taking place. This resulted in air enriched with quanta load not being removed sufficiently quickly. At the same time, it can be seen that an increased local age of air is evident along the entire length of the restaurant, but due to the airflow pattern, infection only occurred in the left area.

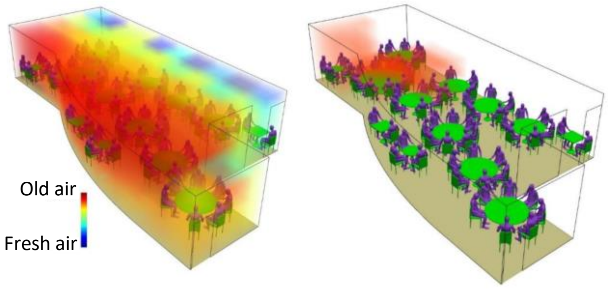


Figure 6: Simulation of the documented case: age of air (left) and increased quanta load in red (right)

With the help of the VEPZO model, an alternative ventilation pattern avoiding mixture of air was investigated to keep the dispersion area of the infectious aerosols more locally confined. Figure 7 shows the modified ventilation where fresh air is supplied centrally at the ceiling along the entire length of the restaurant and exhausted at the lower edges of the room. An improvement in the age of air and particle dispersion paths is clearly visible (Figure 8). With this ventilation arrangement, it is assumed that the number of infected other guests would have been lower.

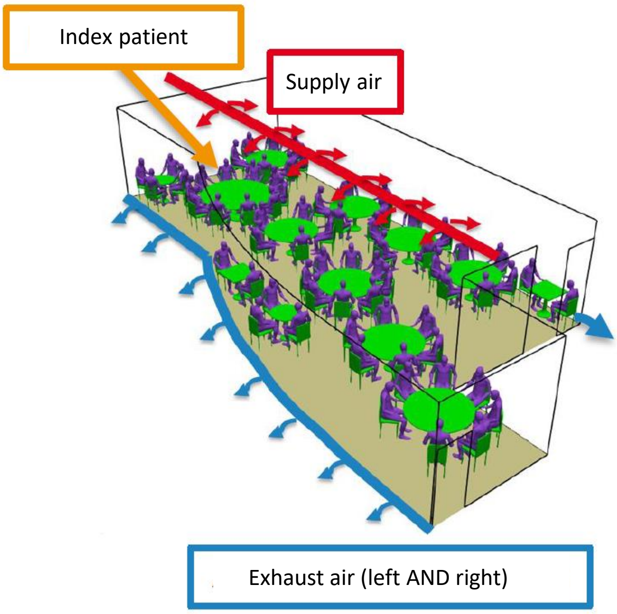


Figure 7: Schematic view of an optimum ventilation system for the considered case found with the VEPZO model

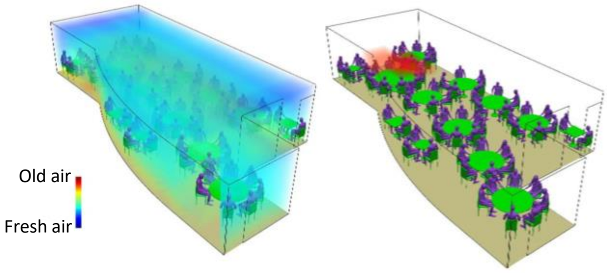


Figure 8: Simulation of the optimised case: age of air (left) and increased quanta load in red (right)

6 . Conclusion

It is shown that the VEPZO model allows quick predictions of the local age of air and the dispersion of compounds such as virally loaded aerosols in indoor spaces. In particular, this simulation uses the notion of quanta to represent Sars-CoV-2 loaded aerosols that remain for longer time within air. Computations converge in the order of minutes on a normal PC and thus the burden to conduct such simulations is considerably lower using the zonal modelling approach than classically applied CFD methods.

For a final risk assessment, the duration of a person's stay in such a particle cloud needs to be taken into account, as well as the actual accumulation of infectious viruses in the air due to various expiratory activities of the infected person (breathing, coughing, sneezing, singing) and the vulnerability of the inhaling person. Nevertheless, a verified method is presented by which the potential risk of infection with Covid-19 and corresponding super spreading events can be predicted by means of zonal models. Further investigations can be based on this study and improvement measures can be evaluated for their effectiveness in a short time in a product-neutral or product-specific manner.

References

Buonanno, G., Stabile, L., and Morawska, L., (2020), “Estimation of airborne viral emission: Quanta emission rate of SARS-CoV-2 for infection risk assessment,” Environment International 141, doi:10.1016/j.envint.2020.105794

Clean Sky JU (2011). Available online at https://cleansky.eu/, updated on 2020, checked on 1/14/2021.

Goscé, Lara; Barton, David A. W.; Johansson, Anders (2014): Analytical modelling of the spread of disease in confined and crowded spaces. In *Scientific reports* 4, p. 4856. DOI: 10.1038/srep04856.

Jimenez, J.L., (2020), “SARS-CoV-2 aerosol transmission estimator,” https://docs.google.com/spreadsheets/d/16K1OQkLD4BjgBdO8ePj6ytf-RpP-MlJ6aXFg3PrIQBbQ/edit#gid=519189277

Lindner, Andreas J.M.; Pschirer, Marie; Norrefeldt, Victor; Siede, Markus (2019): Case studies validating a new climate concept for cold galley areas with the DressMAN and the IESS model. In *AST*.

Lu, Jianyun; Gu, Jieni; Li, Kuibiao; Xu, Conghui; Su, Wenzhe; Lai, Zhisheng et al. (2020): COVID-19 Outbreak Associated with Air Conditioning in Restaurant, Guangzhou, China. In *Emerging Infectious Diseases* 71 (15), pp. 841–843. DOI: 10.3201/eid2607.200764

Melikov, Arsen K. (2020): COVID-19: Reduction of airborne transmission needs paradigm shift in ventilation. In *Building and Environment* 186, p. 107336. DOI: 10.1016/j.buildenv.2020.107336.

Norrefeldt, Victor; Grün, Gunnar; Sedlbauer, Klaus (2012): VEPZO – Velocity propagating zonal model for the estimation of the airflow pattern and temperature distribution in a confined space. In *Building and Environment* 48, pp. 183–194. DOI: 10.1016/j.buildenv.2011.09.007.

Norrefeldt, Victor; Grün, Gunnar; van Treeck, Christoph (2013): Use of the VEPZO model to optimize a hybrid ventilation system.

Norrefeldt, Victor; Pathak, Arnav; Lemouedda, Abdellah; Siede, Markus; Grün, Gunnar (2015):

Validation of the zonal thermal model VEPZO/RADZO for cold outside conditions on a business jet mock-up. In *AST*.

Pathak, Arnav; Norrefeldt, Victor; Lemouedda, Abdellah; Grün, Gunnar (2014): The Modelica Thermal Model Generation Tool for Automated Creation of a Coupled Airflow, Radiation Model and Wall Model in Modelica. In : Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden. the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden, March 10-12, 2014: Linköping University Electronic Press (Linköping Electronic Conference Proceedings), pp. 115–124.

Reim, Hanna; Norrefeldt, Victor; Noisten, Peter; Nasyrov, Vladislav; Stratbücker, Sebastian (2015): Export of BIM data to energy simulation tools and more refined zonal models. In *Lake Constance 5D-Conference*.

Sze To, G. N.; Chao, C. Y. (2010): Review and comparison between the Wells-Riley and dose-response approaches to risk assessment of infectious respiratory diseases. In *Indoor Air* 20.

Villafruela, J. M.; Olmedo, I.; San José, J. F. (2016): Influence of human breathing modes on airborne cross infection risk. In *Building and Environment* 106, pp. 340–351. DOI: 10.1016/j.buildenv.2016.07.005.

Yan, Yihuan; Li, Xiangdong; Shang, Yidan; Tu, Jiyuan (2017): Evaluation of airborne disease infection risks in an airliner cabin using the Lagrangian-based Wells-Riley approach. In *Building and Environment* 121, pp. 79–92. DOI: 10.1016/j.buildenv.2017.05.013.

Zhang, Lei; Li, Yuguo (2012): Dispersion of coughed droplets in a fully-occupied high-speed rail cabin. In *Building and Environment* 47, pp. 58–66. DOI: 10.1016/j.buildenv.2011.03.015.

Model-Based Development of the RespiraWorks Ventilator with Modelon Impact

John Batteh[1] Lixiang Li[2] Edwin Chiu[3] Ethan Chaleff[4]

[1,2]Modelon Inc., USA, {john.batteh,lixiang.li}@modelon.com

[3,4]RespiraWorks, USA, {edwin,ethan}@respira.works

Abstract

This paper describes the modeling and simulation of the RespiraWorks ventilator in Modelica with Modelon Impact and the Pneumatics Library. Following a brief overview of the RespiraWorks open-source effort in response to COVID-19, details of the pneumatic modeling effort, including the implementation of new components, are provided in support of the model-based development process. The pneumatics models of several different iterations of the ventilator design are shown. Lastly an overview of the model calibration process is provided, and the model results are compared with experimental data collected from the ventilator prototype.

Keywords: pneumatics, ventilator, COVID-19

1 Introduction

The COVID-19 global pandemic has resulted in a shortage of critical medical resources. In particular, there has been an acute shortage of ventilators, especially in developing countries. Ventilators are expensive medical devices typically developed over 5-6 years. At the start of the pandemic, three engineers, Ethan Chaleff, Edwin Chiu, and Elizabeth Hillstrom, started an effort to develop a low cost open-source ventilator. Within two weeks, two prototypes had been built in a small Berkeley, CA garage. Within a month, the group incorporated RespiraWorks, a 501(c)(3) non-profit organization. Today RespiraWorks has grown to over 200 volunteers with a range of expertise in 10 countries (RespiraWorks 2021).

From its inception, the RespiraWorks mission is to radically democratize the ventilator. The team set out to create a full-featured ventilator that is affordable and easy to build in countries with developing economies and low-resource communities. With an open-source, IP-free design, organizations can leverage local resources to help their people. The group was motivated to remove money as a barrier for people to obtain life-saving medical equipment and to shift revenue motivation largely to those manufacturing and delivering equipment to those in need.

Many of the ventilator efforts spawned during the pandemic were focused on providing a “bridge” ventilator that provides temporary support until the patient can be placed on a full feature ventilator, reducing but not eliminating reliance on imported ventilators. The RespiraWorks mission is to build a fully-featured, fully-certified medical device capable of advanced respiratory support for patients who may be on a ventilator for days or weeks and whose benefit endures beyond the current crisis. The team is committed to helping manufacturers around the world build the ventilator and has signed a memorandum of understanding with Foundry M in India to develop and manufacture the ventilator for the Indian market.

The RespiraWorks team faced unique challenges at the start of the effort. To make an impact in the current pandemic, the development process needed to be significantly shortened from the typical 5-6 years. Thus, the team needed a flexible, efficient approach with multiple designs developed in parallel. Ventilators are inherently a physical system which are often designed using a hardware-focused design – testing iteration loop. With an organization with no physical base of operations and no shared workspace and distributed globally, the challenges of such a hardware-intensive distributed effort were immense. Furthermore, supply chains were significantly disrupted due to the pandemic with medical-grade parts in short supply. To mitigate these challenges, the team focused on automotive/industrial supply chains, developed custom hardware and sensor solutions using off the shelf parts, and utilized 3D printing for rapid prototyping.

As part of a coordinated hardware and model-based design process, Modelon engaged with the RespiraWorks team to support their efforts. Modelon donated licenses for their new simulation platform Modelon Impact including the Pneumatics Library (Modelon 2021). A team of engineers from Modelon developed pneumatics models of the various design iterations and provided early feedback on design proposals. Simulation was used extensively, especially early in the design process. Using the browser-based Impact platform, Modelon support enabled members of the RespiraWorks team to quickly access the models and execute them to support design iterations.

The following sections in this paper provide an overview of the ventilator modeling effort, including new components developed to support the system design. Pneumatic system models from several design prototypes

DOI
10.3384/ecp21181471

are described along with analysis results from a range of simulations. Lastly an overview of the calibration process is provided along with a comparison of the calibrated model with experimental data from the design prototype developed for the CoVent-19 Ventilator Challenge.

2 Model Component Overview

This section provides an overview of the key components of the ventilator model built with the Modelon Pneumatics Library (Modelon 2021). Following a description of a few key components, the system model architecture is shown.

2.1 Blower

From the outset, the RespiraWorks team set out to design a full-featured ventilator which did not rely on compressed air, ambu-bag, or mechanical bellows. With medical blowers in short supply and only available at a high cost, the team focused on an automotive/CPAP blower that was readily available through automotive supply chains and able to operate at 1/5 the cost and peak power of a medical blower. Figure 1 shows the X200N 12V DC blower and control board.

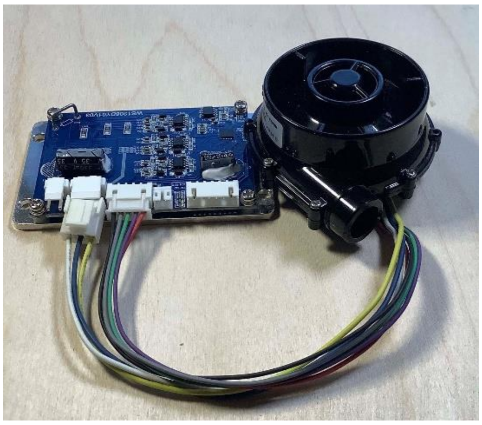


Figure 1. Ventilator blower and control board

A custom blower model was implemented using the fan model from the Modelon library as shown in Figure 2. The fan model includes different options for specifying the flow characteristic based on fan affinity laws. For the X200N model, the table-based characteristic with volume flow rate and power consumption as a function of pressure rise was used. The data provided to characterize the flow is shown in Figure 3 and provides the pressure and volumetric flow but over a range of speeds. Since the table-based model is characterized at a single speed, the table characteristic was tuned to match the data provided resulting in the flow map shown in Figure 4.

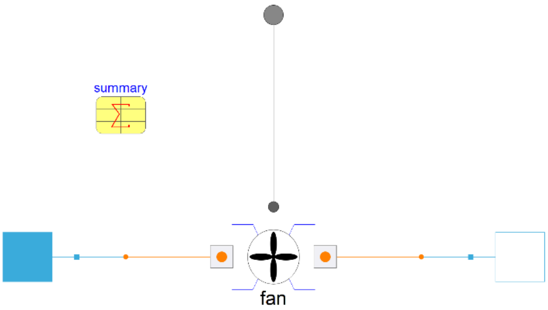


Figure 2. Blower model

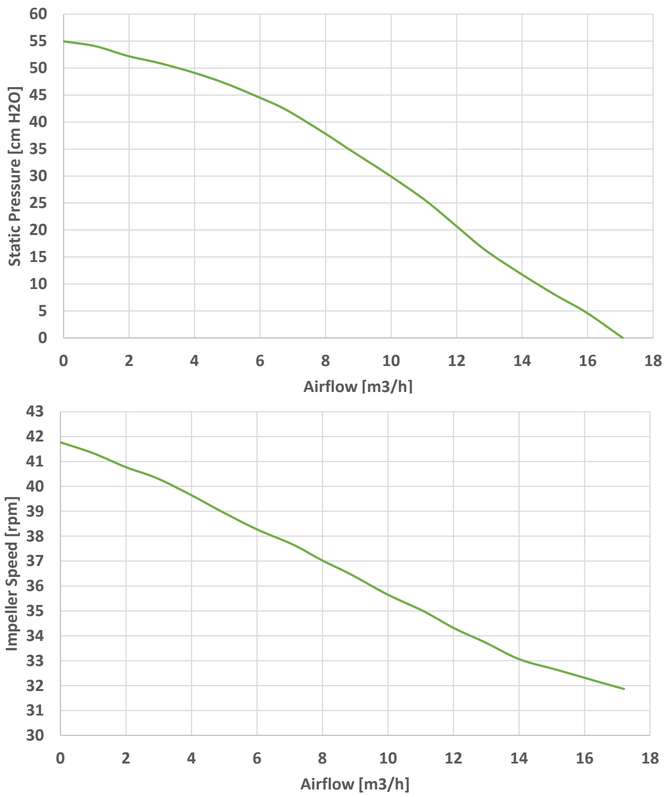


Figure 3. X200N flow characteristics

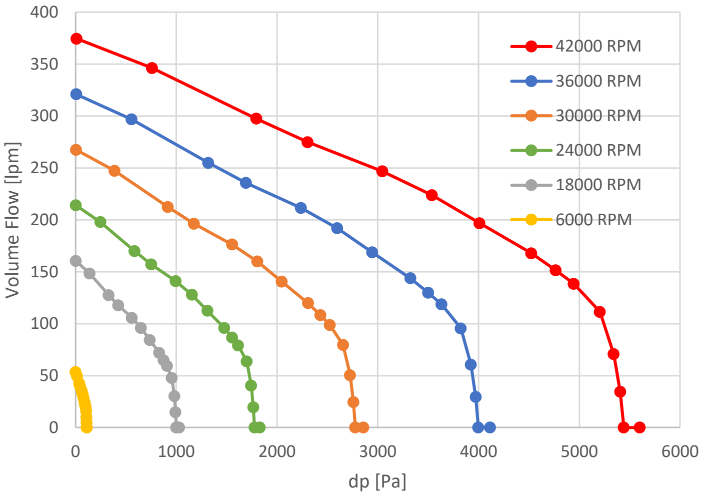


Figure 4. X200N flow map

2.2 Venturi Flow Sensor

Medical flow sensors were completely unavailable at the start of the pandemic. Thus, the RespiraWorks team developed a custom venturi flow sensor using $20 of commonly available automotive parts to replace the roughly $500 cost of unobtainable medical parts. The team arrived at the final design shown in Figure 5 after roughly two dozen iterations. A custom venturi model was implemented to provide the pressure difference based on the characterization performed by the RespiraWorks team. The venturi output pressure difference is provided as a sensor signal to the controller for flow estimation.

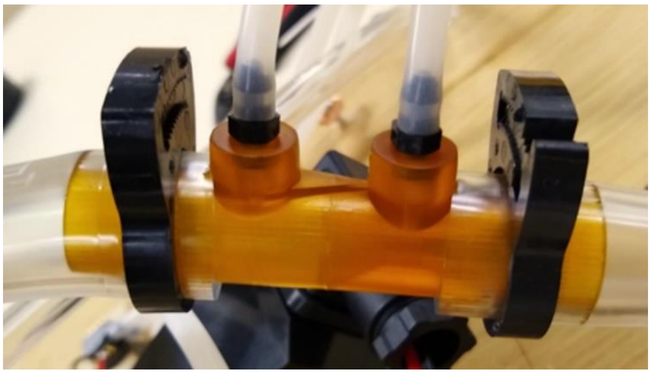


Figure 5. Venturi flow sensor design

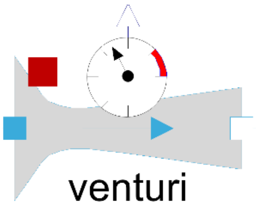


Figure 6. Venturi model

2.3 Pinch Valve

Ventilator designs rely on valves to control the air flow during the patient inhale and exhale phases. Many ventilators rely on solenoid valves. Electrical solenoid valves deliver fast response as required at higher breathing rates but at high cost. In addition, proportional solenoids require a high-pressure source to deliver high flow. Thus, using proportional valves on the air side would also require high power blowers or compressors, or the use of hospital-supplied medical compressed air which is not always available. Early simulation work described in Section 3.1 showed that without a fast, high flow valve, the design would require high peak power capability in the blower to reach specified performance targets.

Thus, the team developed a custom pinch valve shown in Figure 7 for flow control. The pinch valve uses an electrically actuated lever to contract the tube, thereby restricting the flow. Though the pinch valve is not capable of completely closing the flow path, it effectively restricts the flow for practical full range flow restriction. The pinch valve was the key innovation that allowed the team to provide a high-flow fast-response flow control without needing to throttle the blower. It is this combination of fast response and low pressure drop at high flow that enabled the use of a commonly-available CPAP blower as the air-side pressure source and also enabled the 5x reduction in peak blower power by eliminating the need to accelerate and decelerate the blower.

To model the pinch valve, a variable table-based flow model was implemented as shown in Figure 8. This model is based on a flow map for the volumetric flow as a function of pressure drop and opening. Based on data provided by the RespiraWorks team, the pinch valve flow map shown in Figure 9 was developed.

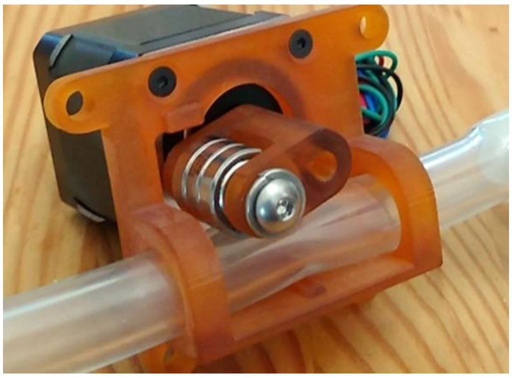


Figure 7. Pinch valve design

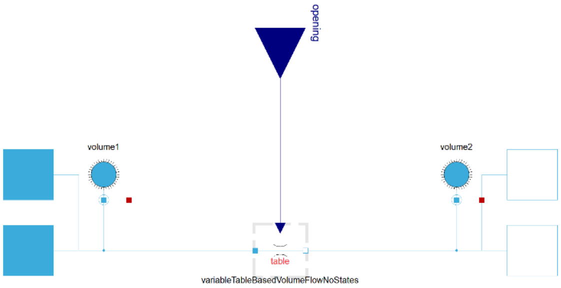


Figure 8. Pinch valve model

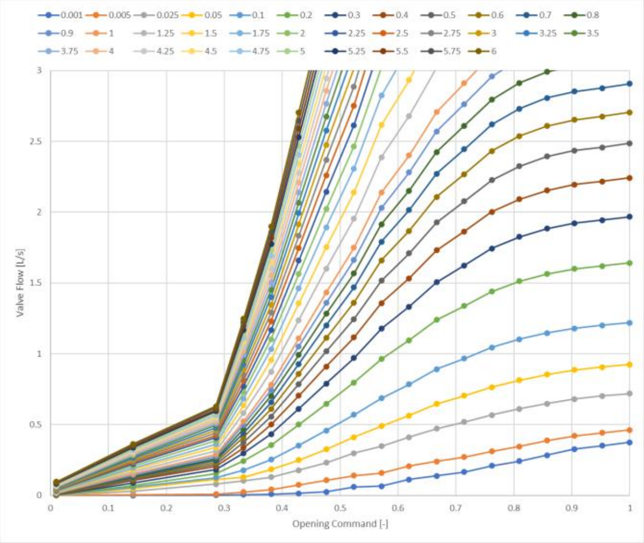


Figure 9. Pinch valve flow map

2.4 Proportional Oxygen Valve

To control oxygen flow to allow a variable ratio of oxygen to air (FiO2), the RespiraWorks team designed a system using a proportional oxygen solenoid valve. Figure 10 shows the characterization data for the PVQ30 solenoid valve. This data was used create a map for the variable table-based resistance model. The map is shown in Figure 11. Note that the hysteresis is not considered in the model as a single average between the opening and closing curves was used.

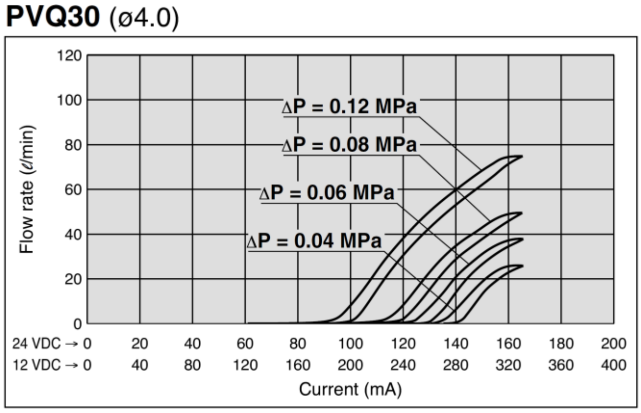


Figure 10. PVQ30 valve characterization data

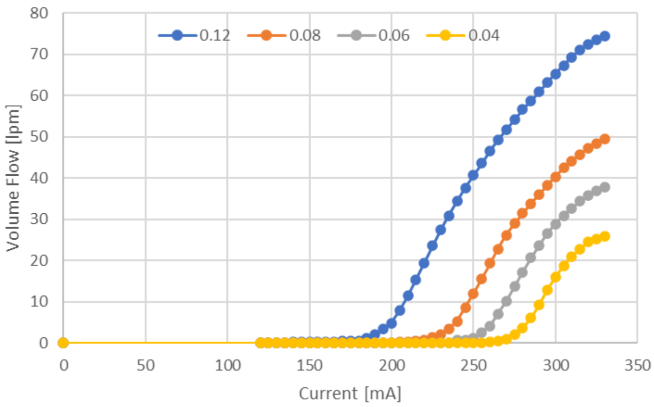
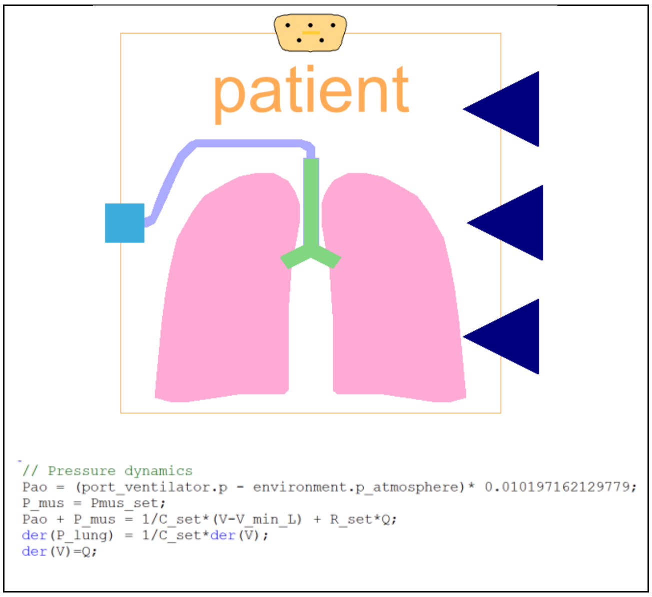


Figure 11. PVQ30 valve data for model

2.5 Patient Model

A model of the patient is a critical need for model-based design of the ventilator. The patient model simulates the breathing mechanics of the patient and can be extremely complicated to account for the various resistances and capacitances in the airway, throat, and lung. In addition, the patient model can include the mechanics of the breathing process to account for an active patient (i.e. a patient that can initiate and breathe either partially or completely). Since full-featured ventilators can operate in different modes and with patients requiring varying levels of breathing assistance, the patient model is an important part of the overall model-based development process.

For this work, a simple equation-based model (Arnal 2018) that is typically used to characterize patients was implemented in Modelica. The model includes an overall lung resistance and capacitance and accounts for an active patient with a musculatory pressure term that acts in conjunction with the pressure at the airway opening provided by the ventilator. The model is shown in Figure 12 and relates the lung pressure, volume, and flowrate. The implementation allows for conditional input for the musculatory pressure and also allows the resistance and capacitance to be set via connectors for dynamic response during a simulation or parameters.



```
// Pressure dynamics
Pao = (port_ventilator.p - environment.p_atmosphere)* 0.010197162129779;
P_mus = Pmus_set;
Pao + P_mus = 1/C_set*(V-V_min_L) + R_set*Q;
der(P_lung) = 1/C_set*der(V);
der(V)=Q;
```

Figure 12. Patient model

2.6 System Architecture

Figure 13 shows the system model architecture developed to support model-based development of the ventilator. It consists of three replaceable subsystems:

- Controller
- Ventilator
- Patient

The architecture allows flexible configuration of the various subsystems including implementation of different controllers to support the various ventilator designs. In addition, the architecture supports unit testing of the various components with simplified implementations (i.e. patient test with ventilator as prescribed pressure trace, etc.). Each subsystem is connected via an expandable controlBus to facilitate configuration of the complete system model and also to allow flexibility in the bus variables for each subsystem implementation. Specific implementations of the various subsystems are detailed in the following section.

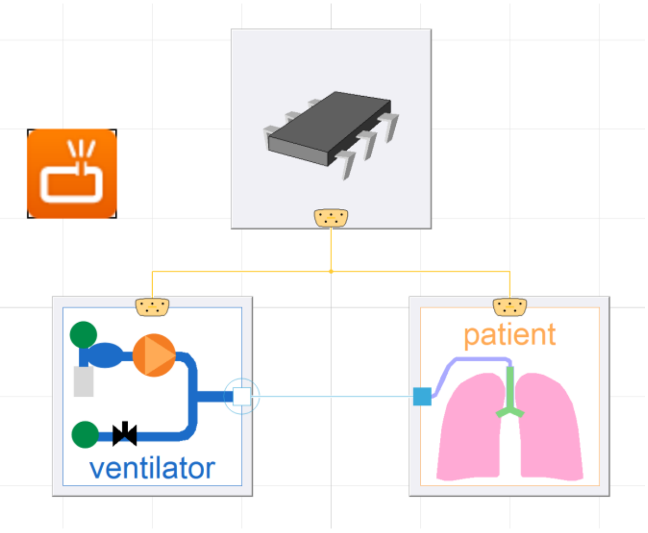


Figure 13. System model architecture

3 Simulation Results

To support the model-based design process of the RespiraWorks ventilator, models were implemented of several different ventilator design prototypes. Analysis results from these models were fed back to the RespiraWorks team to support design iterations. This section provides an overview of several of the ventilator designs, associated simulation results, and key findings from the modeling effort to support subsequent design iterations.

The ventilator modeling and simulation was conducted using the new simulation platform Modelon Impact based on the Pneumatics Library (Modelon 2021). Modelon Impact is a next generation system modeling and simulation platform, leveraging the benefits of web and open standard technologies. With openness at its core, Modelon Impact supports standards such as Modelica, FMI, Python and REST (Modelon 2021). The user-friendly browser interface provides modeling experts the tools they need to create, simulate, and experiment. Steady-state or dynamic simulations can be executed from the same model, reducing effort to get an answer (Coïc 2020b) Finally, the Modelon Impact API enables user-specific workflows through Python-based custom functions, and deployment of models to non-experts via targeted web applications or Jupyter Notebooks (Coïc 2020a).

3.1 Initial Prototype

Modelon engaged with the RespiraWorks team just as the initial system prototypes were being designed and tested. Figure 14 shows a schematic of the initial prototype design focused on basic hardware prove out and controls requirements. Oxygen is introduced upstream of the blower and mixes with air in a mixing chamber. This design was an early attempt at the system design without any active valve control. This design was meant to assess the feasibility of a design concept where all phases of the breathing process were controlled by the blower and a fixed restriction valve on the exhale limb.

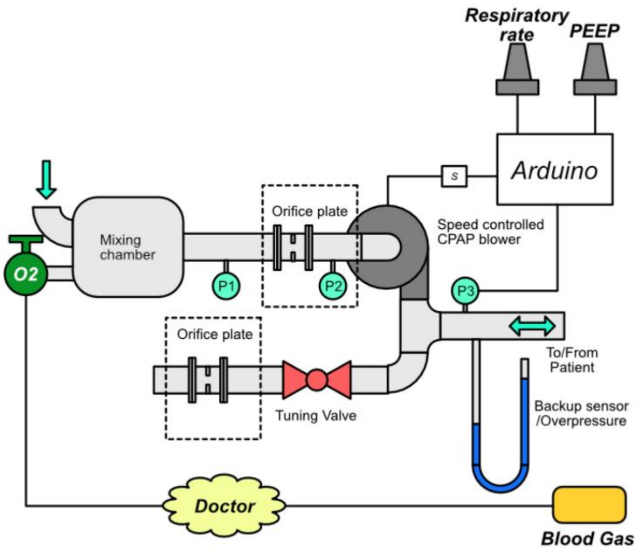
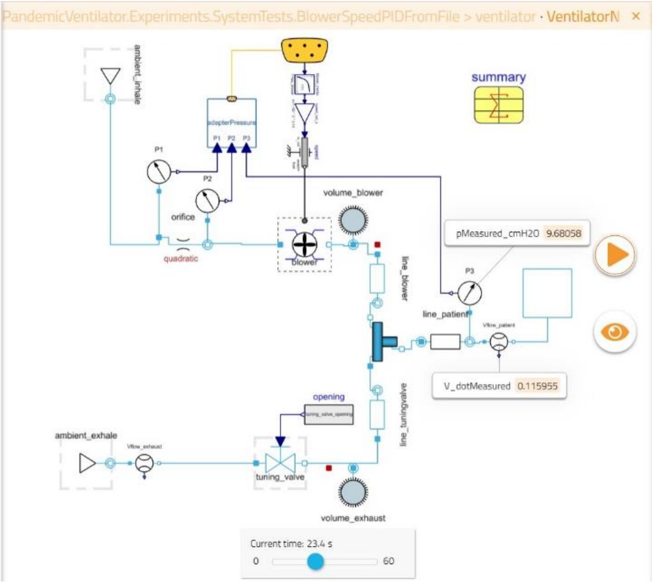


Figure 14. Initial prototype ventilator system

Figure 15 shows the model of the initial prototype in Modelon Impact. The blower speed is controlled to meet a desired pressure P3 at the patient interface as shown in Figure 16 without any pressure sensor dynamics or noise. Note that the controller architecture includes a component for sensor dynamics, but a null implementation of the sensor dynamic component is used in this controller. In this early phase of development, the desired pressure trace was input directly to the model based on sample traces developed for hardware testing.


PandemicVentilator.Experiments.SystemTests.BlowerSpeedPIDFromFile > ventilator · VentilatorN ×

pMeasured_cmH20 9.68058

V_dotMeasured 0.115955

Current time: 23.4 s

Figure 15. Initial prototype ventilator system model

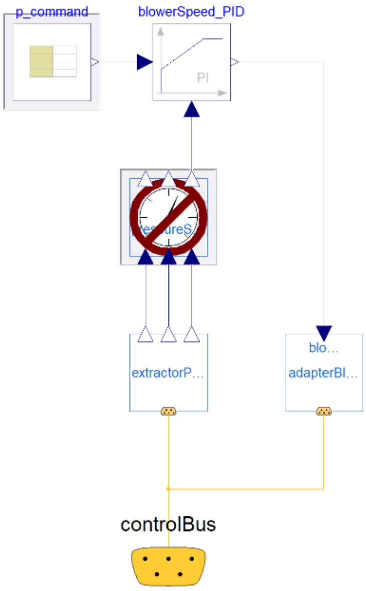
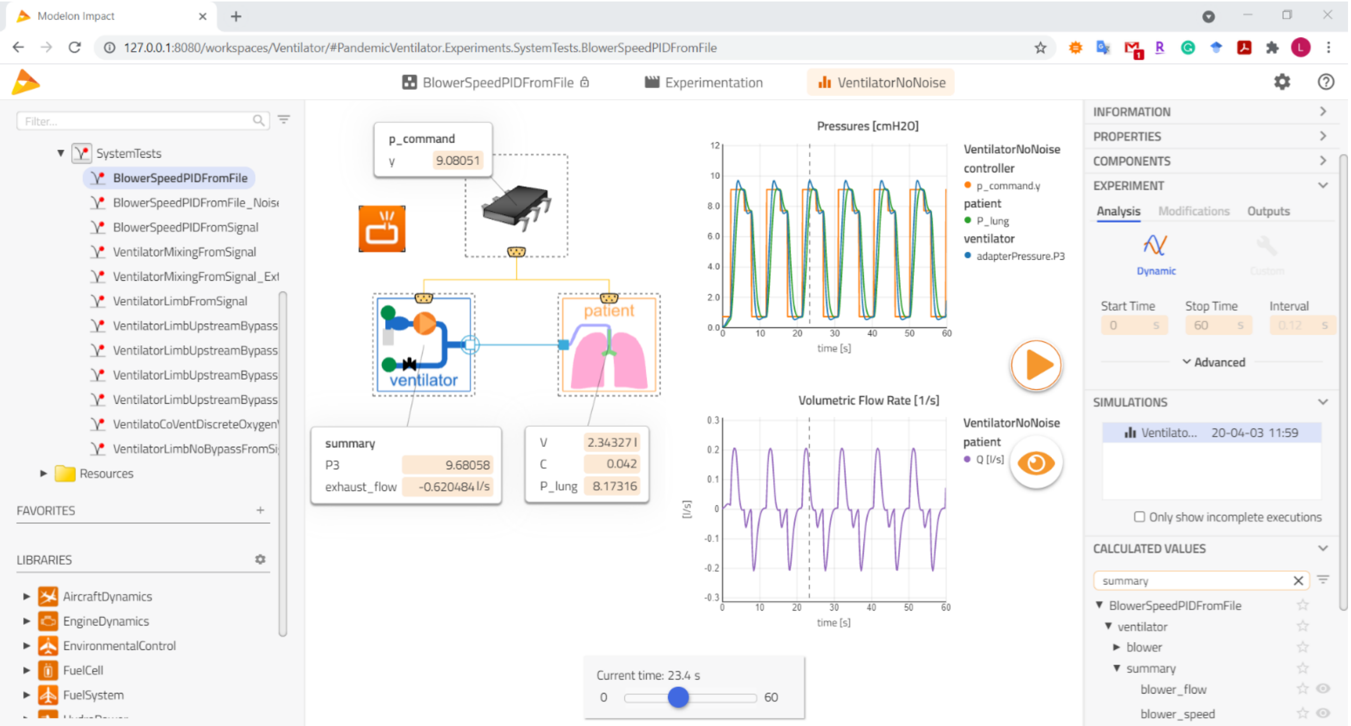


Figure 16. Simple controller for blower speed

Figure 17 shows some initial simulation results integrated with the modeling view while Figure 18 provides a more detailed look at several key results including the pressure response, flowrates, and blower speed. These simulations were run at low breathing rates (~ 6 breaths per minute) and with low pressure targets (9 cm H2O). Based on these initial simulations, the following observations were made:

- The blower speed commands indicate the wide operating range required for a single breathing event
- The blower transient response is a concern and the current design would not meet targets for higher breathing rates
- With the current design, the blower transient response would also prohibit higher pressure targets and thus higher flow rates
- Without control valves on the blower and exhaust legs, there is an enormous amount of oxygen waste and excess blower energy consumption as much of the blower flow flows out the exhaust leg as opposed to entering the patient
- It is possible to tune the exhaust resistance via the tuning valve to reduce the oxygen waste during the intake event but at the detriment to patient exhale
- A fixed resistance on the exhaust limb would likely not satisfy requirements over the full range of operating conditions required for the ventilator

These initial simulation results highlighted the importance of the blower transient response to achieve key ventilator targets and also the need for active control valves. The initial modeling work provided crucial feedback to the RespiraWorks team regarding the blower requirements to meet performance targets and led to the subsequent design iterations, including the development of the pinch valve.


127.0.0.1:8080/workspaces/Ventilator/#PandemicVentilator.Experiments.SystemTests.BlowerSpeedPIDFromFile

p_command
y 9.08051

summary
P3 9.68058
exhaust_flow -0.620484 l/s

V 2.343271
C 0.042
P_lung 8.17316

Start Time 0 s
Stop Time 60 s
Interval 0.12 s

Current time: 23.4 s

Figure 17. Experiment in Modelon Impact

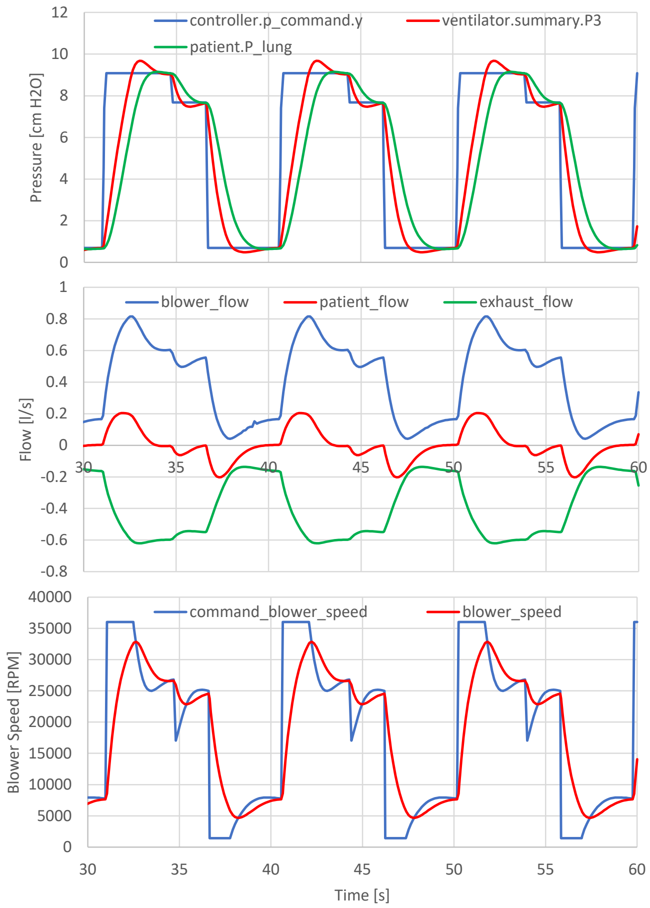


Figure 18. Simulation results for initial prototype

3.2 Ventilator Mixing Concept

Based on the results from the initial prototype, the next design iteration was modeled. They key design changes for the ventilator mixing concept shown in Figure 19 include the following:

- Introduction of oxygen downstream of blower as blower is not rated for pure oxygen flow
- Solenoid on exhaust leg
- Check valves upstream of mixing chamber and on intake leg before patient
- Venturi and filter on both intake and exhaust legs

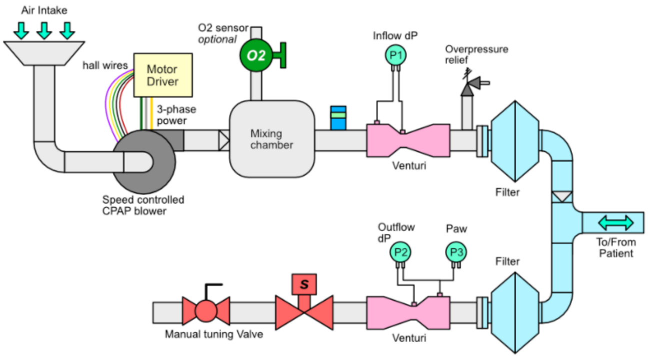


Figure 19. Ventilator mixing concept design

The model for the ventilator mixing concept is shown in Figure 20. This model includes the oxygen source as a prescribed flowrate. The venturi sensors are also included and provide the sensed pressure difference to the control bus. The controller for the model is shown in Figure 21. A custom source block is implemented to provide a pressure command based on parameters for standard ventilator characterization and allows the model to easily run the range of conditions that are required for a full-featured ventilator:

- RR: Respiratory rate in breaths per minute
- IE ratio: Inspiration time to exhalation time ratio
- PIP: Peak inspiratory pressure
- PEEP: Positive end expiratory pressure
- Plateau pressure
- Peak to plateau ratio: Peak pressure time to plateau time ratio

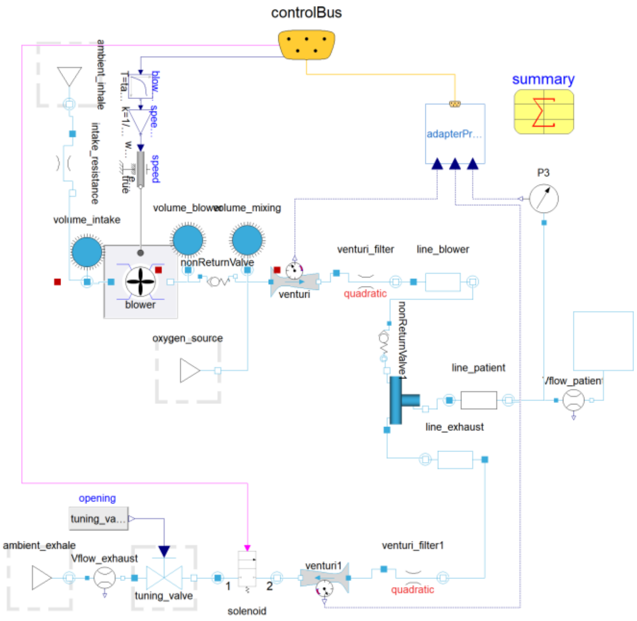


Figure 20. Ventilator mixing concept model

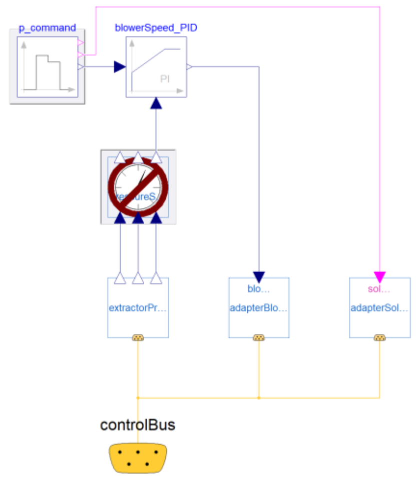


Figure 21. Controller with blower speed and exhaust solenoid based on configurable pressure source

Figure 22 shows simulation results from the ventilator mixing concept with the pressure command settings at RR=20 breaths/min, IE ratio = 1/2, PIP=25 cm H2O, and PEEP = 6 cm H2O for a patient with R = 13 cm H2O s/L, C=0.042 L/cm H2O. These results indicate the following:

- Solenoid eliminates exhaust flow during intake event and reduces oxygen waste
- Even with relatively slow blower transient response, ventilator is just able to meet PIP target at higher breathing rates since exhaust is closed during intake event
- Large flow when solenoid opens (roughly equal to blower flow as blower speed has not reduced plus flow out from dead exhaust volume) helps reduce pressures and facilitate exhale
- Excess exhaust flow from intake (blower + oxygen) until blower speed drops with blip at start of exhaust event due to reduced back pressure
- Blower speed increases around middle of exhaust event to maintain PEEP level

These simulations indicate that the design is significantly improved from the initial prototype and highlight the importance of coordinated valve control along with pneumatic system design.

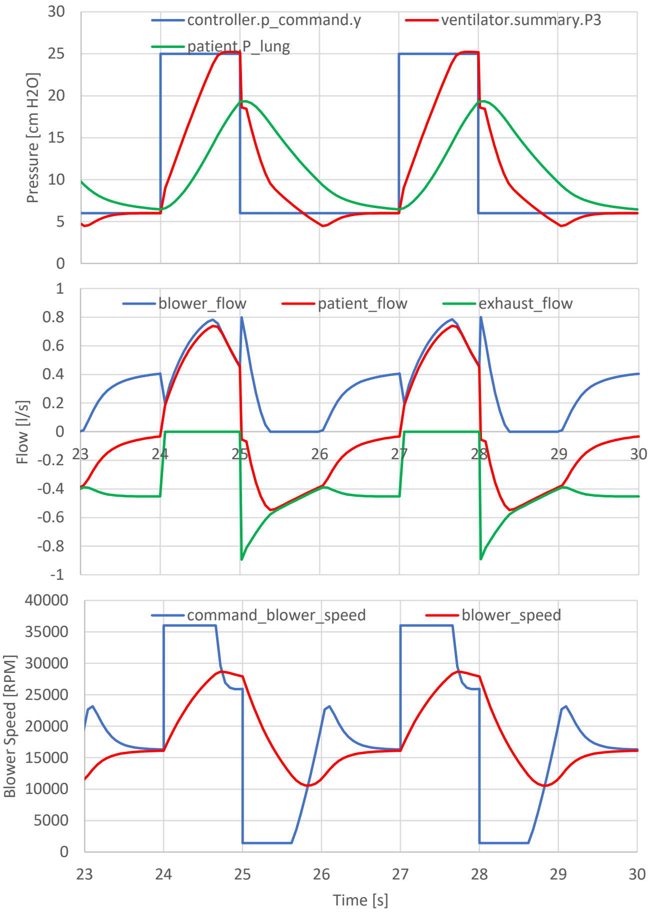


Figure 22. Simulation results for ventilator mixing concept

3.3 CoVent Concept

The next design modeled was the CoVent concept design shown in Figure 23. This design represents design intent for the CoVent-19 Ventilator Challenge. The design updates include the following:

- Addition of closed loop oxygen control via the oxygen proportional solenoid valve and pressurized oxygen source
- Intake blower and exhaust pinch valves

This concept provides full control of the air, oxygen, and exhaust flows via the various valves and allows for closed loop control of oxygen to meet a range of oxygen to air ratios.

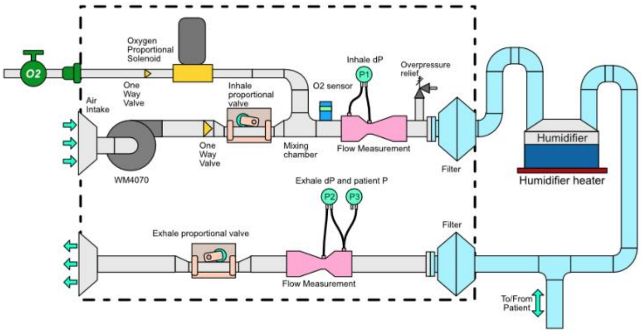


Figure 23. CoVent concept design

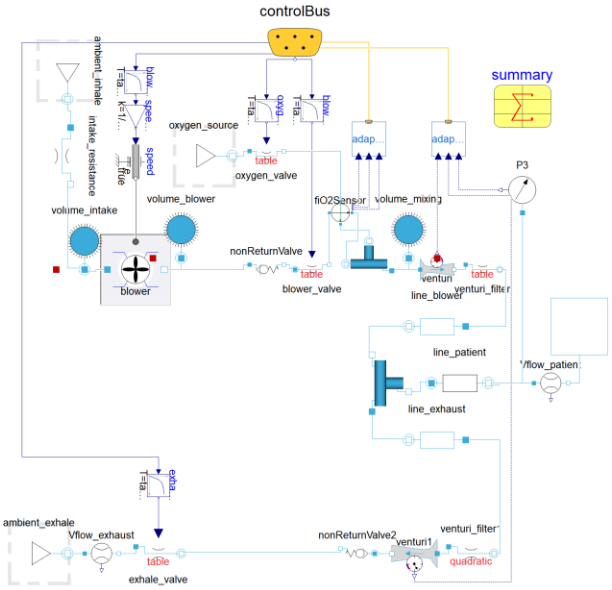


Figure 24. CoVent design model

Along with the modeling effort, the hardware was being developed for testing to support the CoVent-19 Challenge submission. Figure 25 shows the hardware realization, including the QuickLung to simulate the patient. Data was taken on the hardware over the range of operating conditions outlined in green in Table 1. These tests were run with air only and also with oxygen only as the closed loop FiO2 controller was not complete when the tests were run. This data was made available for model calibration. The data includes the pressure control setpoint, recorded pressure, inhale and exhale venturi pressure difference, inhale and exhale pinch valve command, and oxygen valve command.

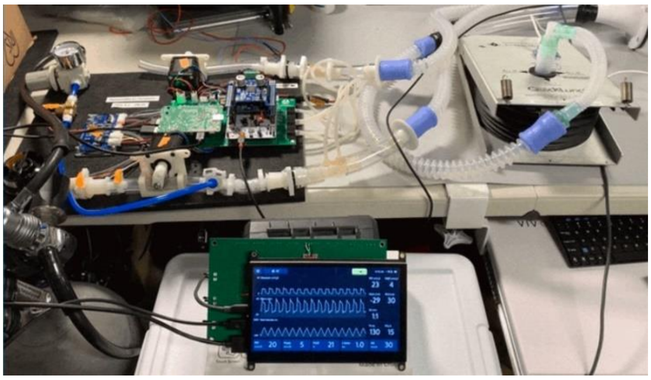


Figure 25. CoVent hardware realization

Table 1. Test settings for CoVent runs

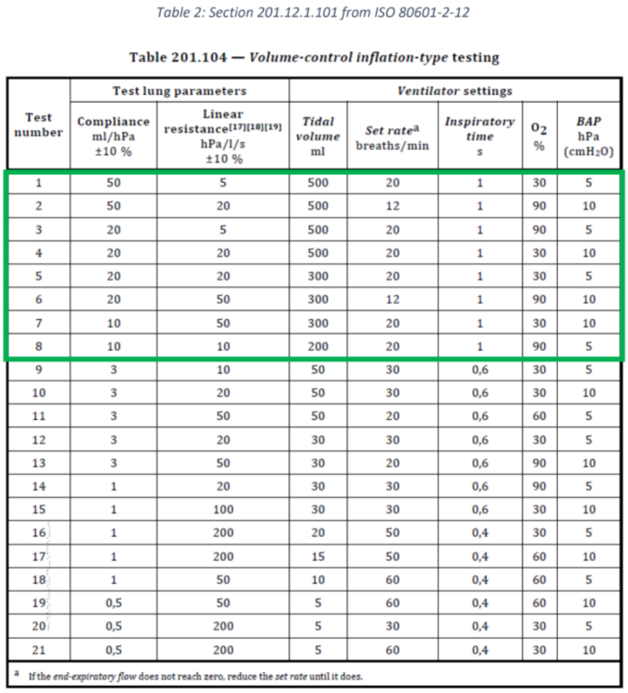


Table 2: Section 201.12.1.101 from ISO 80601-2-12

Table 201.104 — *Volume-control inflation-type* testing

Test number	Test lung parameters		*Ventilator* settings					
	Compliance ml/hPa ±10 %	Linear resistance[17][18][19] hPa/l/s ±10 %	*Tidal volume* ml	*Set rate*[a] breaths/min	*Inspiratory time* s	O2 %	*BAP* hPa (cmH₂O)
1	50	5	500	20	1	30	5
2	50	20	500	12	1	90	10
3	20	5	500	20	1	90	5
4	20	20	500	20	1	30	10
5	20	20	300	20	1	30	5
6	20	50	300	12	1	90	10
7	10	50	300	20	1	30	10
8	10	10	200	20	1	90	5
9	3	10	50	30	0,6	30	5
10	3	20	50	30	0,6	30	10
11	3	50	50	20	0,6	60	5
12	3	20	30	30	0,6	30	5
13	3	50	30	20	0,6	90	10
14	1	20	30	30	0,6	90	5
15	1	100	30	30	0,6	30	10
16	1	200	20	50	0,4	30	5
17	1	200	15	50	0,4	60	10
18	1	50	10	60	0,4	60	5
19	0,5	50	5	60	0,4	60	10
20	0,5	200	5	30	0,4	30	5
21	0,5	200	5	60	0,4	30	10

[a] If the *end-expiratory flow* does not reach zero, reduce the *set rate* until it does.

To facilitate running the model based on experimental data for calibration and validation, the controller shown in Figure 26 was developed. This controller reads the time traces from the experimental data and provides them for input to the model. In this design, the blower is always run at max speed and the pinch valves are used to control the intake and exhaust events along with the oxygen valve. The valve commands are output from the experimental data and used to drive the simulations. After running the initial simulations driven by the experimental data, it was readily apparent that the valve commands did not match with the characterization data provided for the valves. After consulting with the RespiraWorks controller development team, the source of the difference was identified as a feature in the controller that performs a system calibration procedure to identify the min and max operating point for the pinch valve and oxygen valve. In addition, the controller includes a linearization table that maps the valve command to the characterized valve position shown in Figure 9 for a linear response in flow. These tables were also implemented in the controller shown in Figure 26 but were allowed to change in the model for calibration purposes.

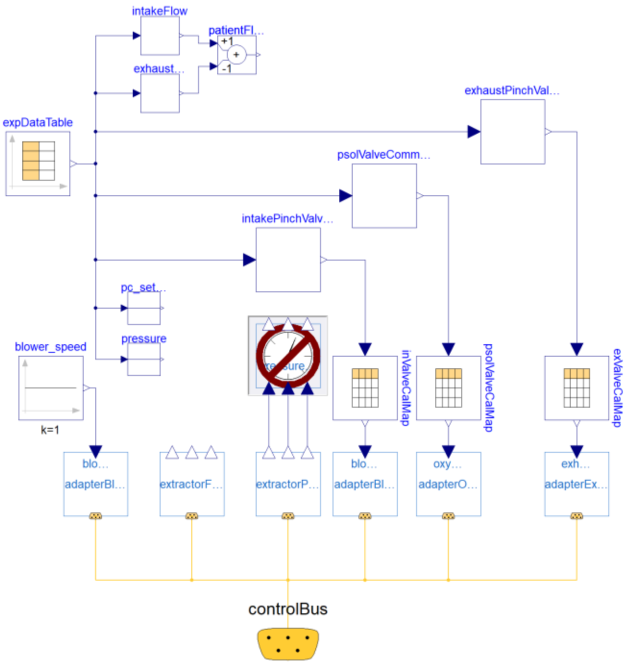


Figure 26. Controller for calibration/validation

The overall calibration procedure for the model is as follows:

- Run model with input flow rate and exhaust valve active to tune overall system resistance and exhaust pinch valve mapping
- Run model with all valves active and adjust blower pinch valve and oxygen valve mapping
- Adjust time constants based on dynamic results
- Re-run all tests for validation

The goal of this calibration procedure is to produce a model of the ventilator system that accurately reproduces the response from the CoVent hardware and can be used to support controller design and calibration. Note the following regarding the calibration approach:

- The blower map was not adjusted
- System response from open loop controller inputs provided as input to the model are difficult to match

- Differences between the simulation and data will exist, especially at steady state
- Overshoot in pressures will persist since controller inputs are provided open loop
- Calibration goal is to get good match to pressures, flowrates, and overall transient response for a range of operating conditions since closed loop controller would improve pressure tracking

Figure 27 – Figure 31 (included at the end of the paper for formatting reasons) compare simulation results with experimental data recorded from the CoVent setup. Each figure compares flow rates and pressures for a specific test condition described in Table 1. Experimental data is indicated in the legend with the "(exp)" label. In the flow plot comparison, the intake and exhaust experimental flow is compared with the blower flow, oxygen flow, and exhaust flow from the model. Since the experimental data was obtained with air only or oxygen only, one of the modeled flow signals will be zero for a given test. In the pressure plot comparison, the pressure setpoint is shown along with the experimental patient pressure from the QuickLung and the model equivalent pressure labeled P3.

Figure 27 shows simulation results from Test 1 with input air flowrate after tuning the exhaust pinch valve mapping and overall system resistance (i.e. components for which no flow characterization was provided). There is excellent agreement in the pressure response when the flowrate is provided as an input in the calibrated model. The results provide validation that the venturi calculation, exhaust valve mapping, and overall system resistance are appropriate.

Figure 28 – Figure 31 show results from different tests in Table 1 with oxygen and air. In these tests, the full model predictions for flowrates and pressures are exercised based on the modeled controller shown in Figure 26. These tests are run with input traces from the experimental data for the intake pinch valve command, oxygen valve command, and exhaust pinch valve command in the modeled controller with resulting model flows.

In general, the results show good agreement with the experimental data. The following observations can be made:

- Overall response for oxygen only runs looks reasonable
- Hysteresis effects (Figure 10) from the oxygen valve are seen in data but not in model as model overpredicts flow decrease during closing command
- Overall response for air only runs looks reasonable
- Intake flow overpredicted a bit which could be attributed to the blower map
- Model results are consistent with higher flowrates resulting in pressure overshoot

4 Summary

This paper describes the model-based development of the open source RespiraWorks ventilator. Using the Modelon Pneumatics Library in the Modelon Impact platform, various design iterations of the ventilator were modeled. The ventilator designs were tested in a configurable system architecture in conjunction with controller implementations and a model of the patient. The RespiraWorks ventilator design for the CoVent-19 Ventilator Challenge was modeled and calibrated using experimental data collected to support the challenge submission. The calibrated model showed good agreement with experimental data. Future work with the model will focus on the application of the model for controller design and tuning.

The RespiraWorks team finished 3rd in the CoVent-19 Ventilator Challenge and is continuing their effort to design and build their ventilator. For the latest updates on the design process, visit the repository at `https://github.com/RespiraWorks/Ventilator`

Acknowledgements

The authors would like to thank Anand Pitchaikani and Midhun Joy from Modelon for their contribution to the model development and Impact deployment at RespiraWorks. The authors also extend their sincere thanks to the entire RespiraWorks team, particularly David Gershon, for his engagement with the modeling effort.

References

Arnal, Jean-Michel, Aude Garnero, Mathieu Saoli, and Robert Chatburn (2018). "Parameters for Simulation of Adult Subjects During Mechanical Ventilation". In: *Respiratory Care*. February 2018. Vol. 63. No. 2. pp. 158–168. DOI: 10.4187/respcare.05775 .

Coïc C., J. Andreasson, A. Pitchaikani, J. Åkesson, and H. Sattenapalli (2020), "Collaborative Development and Simulation of an Aircraft Hydraulic Actuator Model", *Asian Modelica Conference*, Tokyo, Japan.

Coïc C., M. Hübel, and M. Thorade (2020), "Enhanced Steady-State in Modelon Jet Propulsion Library, an Enabler for Industrial Design Workflows". *American Modelica Conference 2020*, Boulder, Colorado, USA,

Modelon (2021). *Impact*. URL: https://www.modelon.com/modelon-impact/.

Modelon (2021). *Pneumatics Library*. URL: https://www.modelon.com/library/pneumatics-library/.

RespiraWorks (2021). URL: https://www.respiraworks.com.

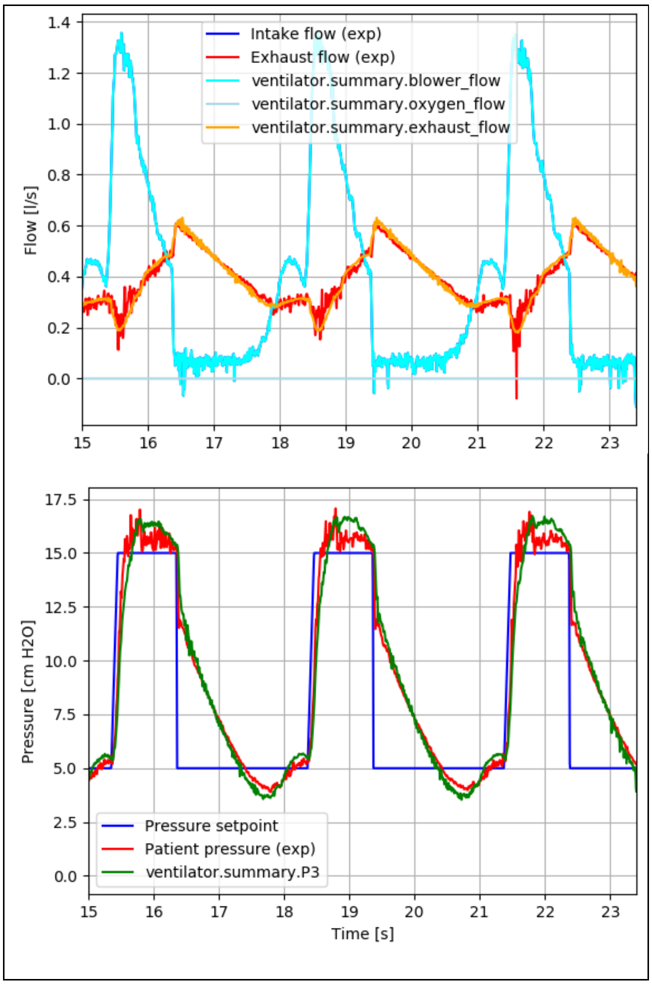


Figure 27. Simulation results from Test 1, air only, input flowrate

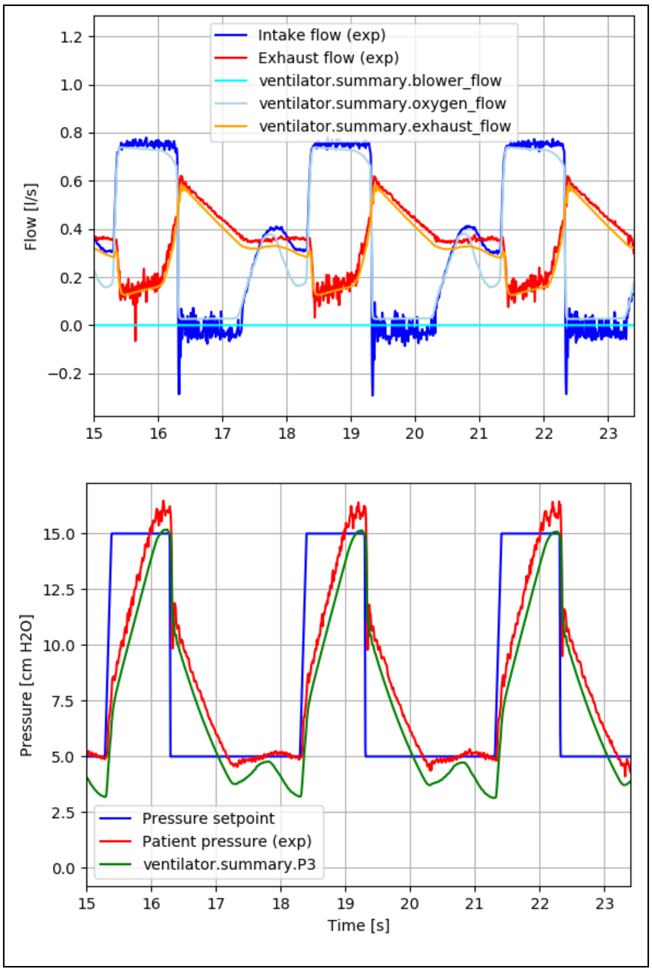


Figure 28. Simulation results from Test 1, oxygen only

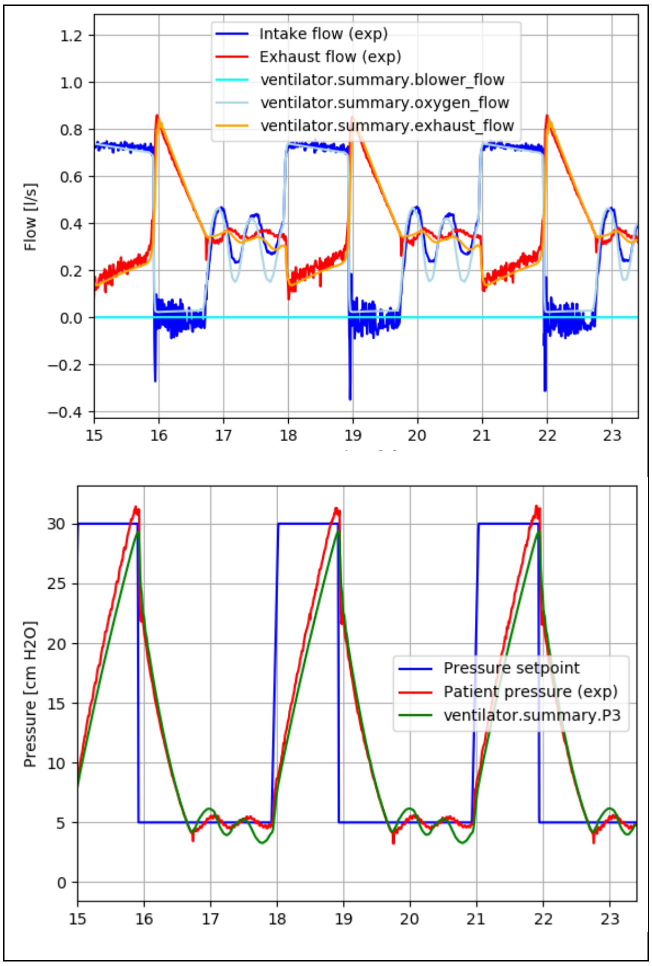


Figure 29. Simulation results from Test 3, oxygen only

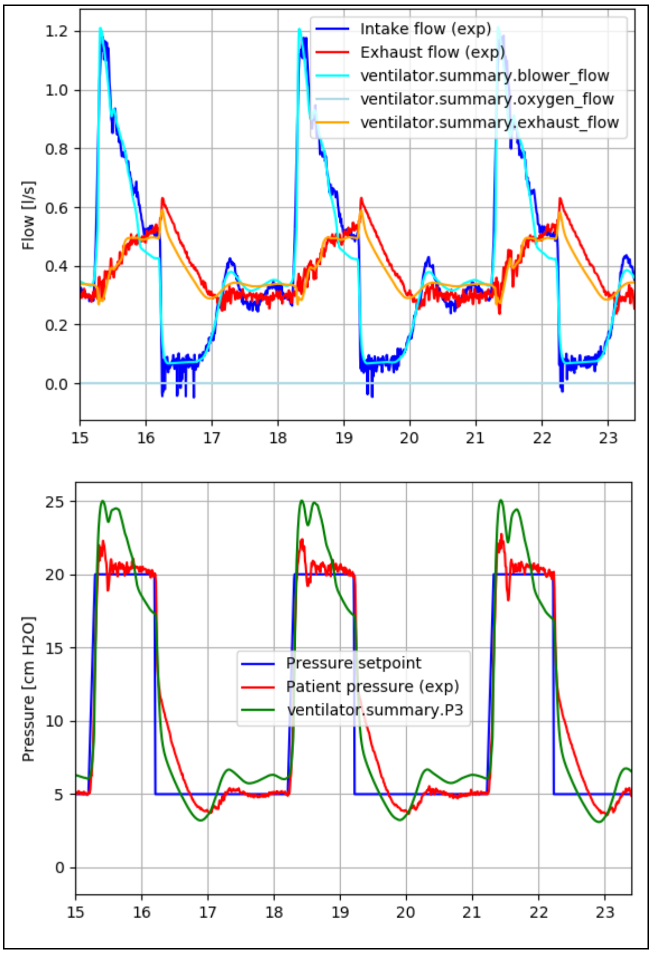


Figure 30. Simulation results from Test 5, air only

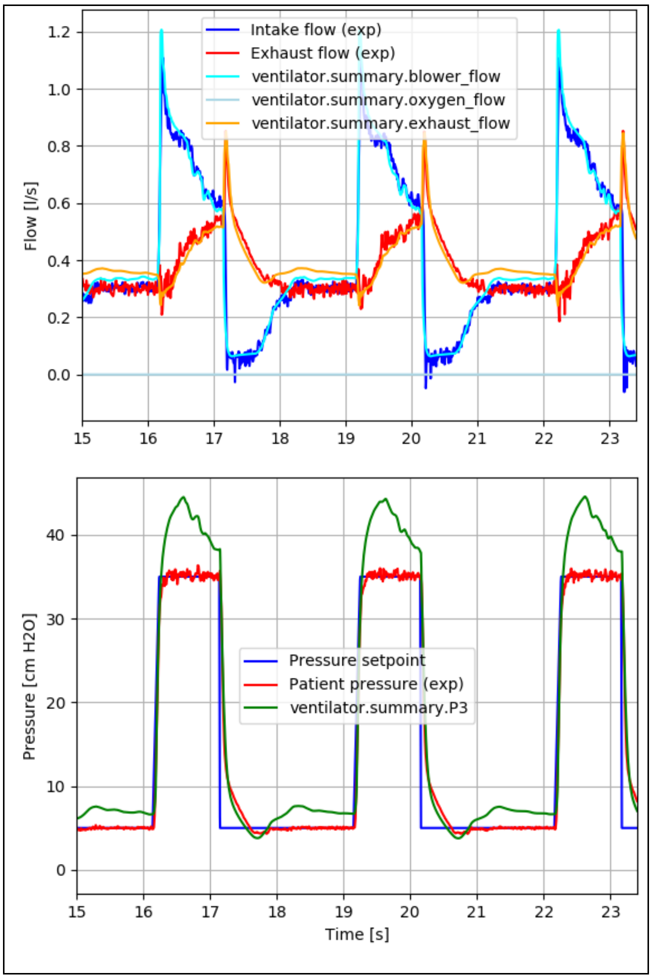


Figure 31. Simulation results from Test 7, air only

In-silico virtual prototyping multilevel modeling system for Cyborgs (CybSim) as a novel approach for current challenges in biosciencies

Manuel Prado-Velasco

Department of Graphic Engineering and Multilevel Modeling in Bioengineering Group, University of Seville, Spain,
mpradov@us.es

Abstract

There is a lack of Modeling and Simulation software systems in the bioscience arena that give both solutions compliant with current methodologies in drug discovery (pharmaceutic) and precision medicine (healthcare) fields, besides to support the addition of new biological mechanisms under a multilevel and multiformalism perspective, without penalize strongly the model sharing and reusing. A novel modeling and simulation software that tries to fill the previous gap has been designed (CybSim) and it is presented in this work. CybSim is a platform for multilevel modeling of physiological - cybernetic systems, compliant but not limited to Physiologically based-, Pharmacokinetic and Pharmacodynamic (PBPK/PK/PD) methodologies. This capability is governed through the *Physiological Scope* setting value. The main physiological components are mechanistic. The underlying mechanisms may be changed during the model building thanks to the separation between mechanisms and physiological instances. This capability is based on a *multi-layer design*. A preliminary version of CybSim has been implemented with OpenModelica (v1.14.1). A PBPK semiphysiological model published previously has been built as a case study to demonstrate the feasibility of CybSim. The accuracy of CybSim was verified during preliminary development phases. The two pointed out capabilities of CybSim demanded an object-oriented and acausal equation-based modeling language, able to support classes' redeclaration, connectors' causality, inner/outer scoping control and packages organization. These features are not supported by other modern acausal equation-based modeling languages like the EcosimPro language.

Keywords: Cyborgs, Physiological modeling, PBPK, Mechanistic Modeling, acausal equation-based Modeling

1 Introduction

The field of modeling and simulation in biosciences is a mature domain that joins efforts from many areas, including biomedical engineering, mathematical biology, and pharmacology. Two big projects that started at the end of 1990's and 2000's may be cited as reference efforts in bioscience modeling. The Physiome initiative was presented in a report from the Commission of Bioengineering in Physiology to the International Union of Physiological Sciences (IUPS) council at the 32nd World Congress in Glasgow (UK) in 1993 (Hunter 2006, Box 1). The Virtual Physiology Human (VPH) project was initiated by the European Commission in 2007, after the publication of the Strategy for a European Physiome (STEP) (Hunter and Viceconti 2009). Both initiatives share as ultimate goal the research and development in computational models, data and tools for a better comprehension of human body under an integrated approach. Many projects defined under the umbrella of Physiome and VPH have pushed the development of a software framework for building mechanistic mathematical multiscale models from cellular to organ levels, with clinical, pharmacological and scientific applications.

A primary objective of VPH - Physiome initiatives was the establishment of model standards and repositories for which the well-known Extensible Markup Language (XML) was selected as a basis approach. Two relevant examples of modeling standards are CellML[1] and SBML[2]. CellML was created for the modeling of cell - level dynamics, whereas SBML is a modeling language for networks of biological compounds (e.g. metabolic pathways) described using a systemic approach (Systems Biology). SBML and CellML encode models, metadata, and data, which represent the cited spatial scales (levels) of the physiological system to be modelled, in a robust and accurate manner. However, CellML and SBML models must be linked during the numerical integration to simulate the full system. The modeling and simulation tool that perform this task is an Achilles' heel in this process, because it should be compliant with CellML and SBML and other standard modeling languages that represent the remaining physiological levels (tissues, organs, and living system). An interesting example is the Cardiac Physiome Project shown in Hunter and Viceconti (2009, Figure 6). In practice, the integrated framework must manage the interaction among models executed in different tools (Sauro et al. 2004). This approach is limited to interacting models that represent systems with weak coupling. On

[1] www.cellml.org

[2] www.sbml.org

DOI
10.3384/ecp21181485

the contrary the accuracy and stability of the full model is degraded due to the jacobian deformation.

The increasing complexity of biosciences compels the model standards to evolve. For instance, the SBML v3 standard tries to give response to the requirements of new mathematical methods and physiological methodologies (Keating et al. 2020). This is a second Achilles' heel of XML modeling languages, since mechanisms and approaches are restricted to fulfil those data formalisms and levels.

As a consequence, although the SBML, CellML and others XML based modeling languages are designed to build multiscale and modular models under a reusable context, the two aforementioned process limitations difficult the model reusing.

Due to the population requirements of the pharmacology industry, the Non Linear Mixed Effect (NLME) models have guided the building of models in this field (Bonate 2011). In addition, Pharmacokinetics (PK) and Pharmacodynamics (PD) methodologies have defined during decades the basis of the deterministic block of NLME models due to their simplicity and success in the description of drug distribution and clinical responses. The PK methodology has evolved to the Physiologically based - PK approach (PBPK) to consider anatomical and physiological features and to improve the predictive ability of NLME models, which is required to address with Drug-Drug interactions (DDI) and special or vulnerable populations, for which clinical trials are not suitable (Jones, Gardner, and Watson 2009). As a consequence, most of the mature commercial software tools on pharmacology industry are based on NLME with PK/PD (NONMEM (2021)), or PBPK/PD (SimCyp (2021), GastroPlus (2021), Open-Systems-Pharmacology (2021)).

The gold standard in Population PK/PD modeling, NONMEM, has driven a different approach to achieve the model reusing and sharing in pharmacological sciences. Despite NONMEM models are built through their ordinary differential equations (ODEs), the wide diffusion of NONMEM, R mathematical software[3] and SBML, promoted the development of a new model standard, PharmML, with the aim of promoting the model sharing. A PK/PD model developed in PharmML is managed by a compliant software tool that may also import SBML code and convert the final model to NONMEM, R, SymCyp, and other well-known modeling software tools (Bizzotto et al. 2017).

The pharmacology industry has pushed the acceptation and evolution of physiological models in the framework of the Quantitative Systems Pharmacology (QSP), which is an approach to translational medicine that combines computational and experimental methods to the development and use of molecules and biologic drugs at the beginning of 2000's (Azer et al. 2021). Many advances in QSP Modeling are supported by PBPK/PK/PD-based NLME approaches with increasing efforts to facilitate the inclusion of new knowledge discovering and modeling strategies from Physiome - VPH projects. However, PharmML does not give a solution for the model sharing and reusing requirement in QSP.

A different strategy to achieve the model reusing comes from modeling languages based on equations' formalisms that do not depend on the algorithmic causality (Roa and Prado 2006, Figure 8). This approach has proved its feasibility in the engineering field with EcosimPro language (EL) (Empresarios Agrupados 2019) and Modelica (Fritzson 2015) as two cutting edge object-oriented (OO) and acausal equation-based modeling language references. EL is a proprietary language implemented in the EcosimPro software tool from Empresarios Agrupados Internacional (EAI), whereas Modelica is a freely available language from the Modelica Association, implemented in many open and proprietary software tools. This modeling approach has been hardly applied in the biosciences arena so far.

Physiolibrary is a library of specialized Modelica components for the building of complex physiological models (Mateják et al. 2014) that tries to give a solution for the lack of adoption of Modelica in biosciences. It emerged from the construction of a large model of human physiology, Physiomodel, which in turn is an extended Modelica version of the integrative human physiological model called HumMod (Hester et al. 2011). The recent study of Ježek et al. (2017) describes a methodology for creating cardiovascular system models with different complexity based on Physiolibrary with the main objective of demonstrating the feasibility of a standardized platform for model reusing. They show an interesting model that considers the complex interactions between cardiac circulation and arterial systems, founded on a hierarchy of subsystems that takes advantage of the encapsulation and acausal equation-based nature of Modelica. However, neither Physiolibrary (Mateják et al. 2014) nor the derived cardiovascular system model (Ježek et al. 2017) offer a modeling and simulation software tool oriented to the specific requirements and challenges in biosciences. For instance, it is not an easy task to adapt any of them for the solution of parameterized population PBPK models that predict the distribution, therapeutic response and potential interactions of a drug, which is a current standard problem in the pharmacology area.

PhysPK is a software tool for modeling and simulation in biosciences implemented with EL that was designed to fill the gap between open and specialized tools in PBPK/PK/PD that offers multilevel model reusing (Prado-Velasco 2016). It was created thanks to an agreement between EAI and me (2015 - 2018) in which I (intellectual owner) worked as team leader, designer and main developer. Several studies have shown the feasibility and accuracy of PhysPK to develop population PK and PBPK models, bioequivalence analysis, and even to generate predictive engines for precision medicine (Reig-Lopez et al.

[3] www.r-project.org/

2020; Gonzalez-Garcia et al. 2017; Prado-Velasco, Borobia, and Carcas-Sansuan 2020). However, the extension from PBPK to other modeling approaches is limited through the change of the input/output role in selected variables during the translation process, what induces numerical problemas in non-desired flow-pressure transients (any PBPK model in PhysPK is a cardiovascular model). The impossibility to associate chemical names to the enumerated values and the difficulty to manage causal blocks are additional limitations.

In addition, neither Physiolab nor PhysPK has the capability to select the physiological mechanisms during the model building.

In summary, to the best of my knowledge, modeling and simulation software systems fail to provide a standardized framework for specialized bioscience areas like pharmacology (Maharao et al. 2020), precision medicine (Polasek, Shakib, and A. Rostami-Hodjegan 2019; Darwich et al. 2017), toxicology (Paini et al. 2019; Bloomingdale et al. 2017), and regulatory decision (Rowland, Lesko, and Rostami-Hodjegan 2015; Shepard et al. 2015), with multilevel and evolutive model reusing and sharing capability.

The goal of this paper is to show a preliminary version of a novel multilevel modeling and simulation software for physiological - cybernetic systems (Cyborgs) based on Modelica and implemented with OpenModelica 1.14.1, called Cyborg Simulator (CybSim). It has been designed for the aforementioned biosciences areas with emphasis in model reusing. CybSim models may be built according to different modeling methodologies, including PBPK/PK/PD, and the physiological mechanisms can be selected during the model building, what facilites the model evolution. The diffusion of therapies where a machine is linked to the human body, in a temporal (hemodyalizer) or more permanent manner (artificial heart or insulin pump), is considered through the inclusion of a dedicated machines package. The current presence of computational control and logic in almost all therapy machines explains the selection of Cyborgs as system target. CybSim will be under open source license and available for download.

The work is divided in two stages: a brief presentation of the CybSim design (first) and a study case based on the semiphysiological PBPK model of Mangas-Sanjuan et al. (2018) to demonstrate the feasibility of CybSim (second). The model from Mangas-Sanjuan et al. (2018) was implemented in NONMEM and PhysPK (Reig-Lopez et al. 2020). The experience achieved in that study has facilitated a preliminary and succinct comparison of CybSim against PhysPK. The accuracy of the CybSim PBPK model was verified during the previous development stages.

It is noted that a detailed analysis of more complex physiological models exceeds the scope of this paper, what justifies the use of very single mechanisms in the study case.

2 Methods

The study is divided in two stages. The CybSim design is briefly presented in the first stage, which includes two steps.

1. General perspective of CybSim. It includes the package organization and some main setting properties.
2. Modeling strategy. The concept of multilayer design is explained and associated to the separation between mechanisms and physiological entities. Some design concepts related to the machine and signals packages are also presented in this context.

The second stage develops a study case based on a semiphysiological PBPK - based model. It comprises two steps.

1. Building of a semi-physiological model that includes intestinal lumen, gut, liver, a systemic plasma compartment (central) and a peripheral compartment. A solid form of parent drug (PD) is administered. This drug is metabolized in a principal metabolite (PM) and secondary metabolite (SM). The model is presented succinctly in the study case Section, although a detailed description is available (Mangas-Sanjuan et al. 2018; Reig-Lopez et al. 2020).
2. The CybSim model was executed under a periodic administration of the PD solid form, first with the original liver mechanisms of the reference model (Mangas-Sanjuan et al. 2018), and second after the modification of the molecular binding mechanism in the liver.

The study case tries to show the feasibility of CybSim to support the change of any underlying mechanism of a physiological instance during the model building.

Other aspects of the CybSim modeling exceed the scope of the paper. The accuracy of the model was verified during the building process.

3 CybSim design

3.1 General properties and architecture

Some key issues of the CybSim design are presented in this Section. They referred to the preliminary version 0.2, implemented in OpenModelica 1.14.1. The Figure 1 shows the packages's organization of CybSim. These are grouped as follows:

- Specific units, main properties and simulation modes of CybSim: `SIunits`, `Properties`.
- Connectors and partial classes for the main interfaces: `Interfaces`.
- Packages that manage biological and physiological data: `BioData`.

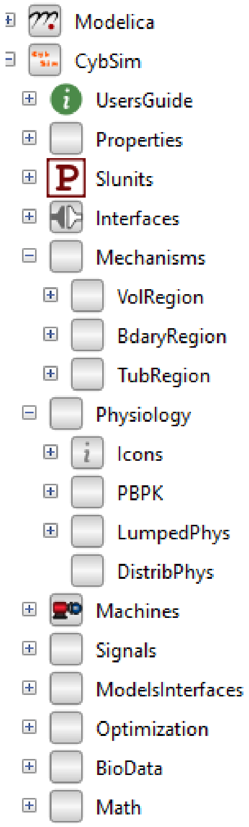


Figure 1. Packages' organization of CybSim v0.2.

- Packages that supports the three CybSim layers: mechanisms, machines and physiological components, and data-driven models: `Mechanisms`, `Machines`, `Physiology`, `Signals`.

- Templates for model applications at different fields: `ModelsInterfaces`.

- Auxiliary packages with mathematical methods and Optimization procedures: `Math` and `Optimization`.

The main features of CybSim are summarized as follows:

- Physiological multiscope. Physiological subsystems in CybSim may be built according to different modeling methodologies, and thus with different scopes. CybSim v0.2 includes PBPK, LPhys and DPhys. A LPhys (general lumped parameter) model is a generalization of a PBPK model where the blood flow rates result from the cardiovascular circulatory system dynamics. Many times a PBPK, where blood flow rates are set by the modeler, is the right choice for a QSP study, and it has lesser computational requirements. The PBPK physiological scope in CybSim is compliant with the PBPK/PK/PD approach addressed in the Introduction. The DPhys refers to a spatially distributed modeling (not yet madure in CybSim). The Listing 1 shows the `PhysScopeType` enumeration type, which controls this feature and it is defined in the properties package.

- Free definition of chemical compounds for models. Chemicals compounds are addressed through a enumeration type (`chemicals`). The chemicalsDummy type (see Listing 1) is assigned as initial chemicals type, which must be redeclared in the final model, as shown in Listing 9. Several key entities in CybSim are defined in a vectorial (array) mode with chemicals as dimension. The physicochemical properties of the chemical compounds may be defined both manually or from a chemical database.

- Chemical volumes. CybSim v0.2 considers three modes of computing the chemicals volume in solution, governed by the `ChemicalVolumeType` enumeration (see Listing 1). The `NoVolume` mode (zero volume) is commonly used for small molecules, whereas the `SmallAsSolvent` considers the true dissolved density of large molecules. This feature requires the discrimination between no large and large molecules. A second enumeration type, `mcrChemicals`, which must be redeclared in the final model, defines the large molecules, whereas `chemicals` includes all of them.

- Multilevel modeling. Multilevel is addressed as synonymous of spatial multiscale. This feature is achieved by means of the aggregation (connection) of physiological components at each level, as shown in the conceptual structure of Figure 2 for the physiology subsystem. The same feature is available for the machines and signals subsystem. This is directly derived from the OO and acausal equation-based property of Modelica and it is available both in textual and graphical model according to the Modelica specification 3.4 (Modelica Association 2017) (MSLv35 has been recently delivered). A key issue here is that the granularity level of different tissues may be different.

- Multilayer architecture. The physical mechanisms that govern the dynamics of any physiological entity are selected during the model building. CybSim achieves this feature through the redeclaration of the inherited mechanisms following the methodology that is explained in the following Section. Conceptually, the physiological layer is defined as a set of physical and mechanistic components that may be defined at different scales (levels) as shown in Figure 2. The mechanism layer is not a physical but an organized set of abstract (partial) components. Although it is not presented in the Figure 2 for the sake of clarity, Machines and Signals may have a multilevel definition and they pertain to Machine and Signals layers, respectively, but they have not a sepa-

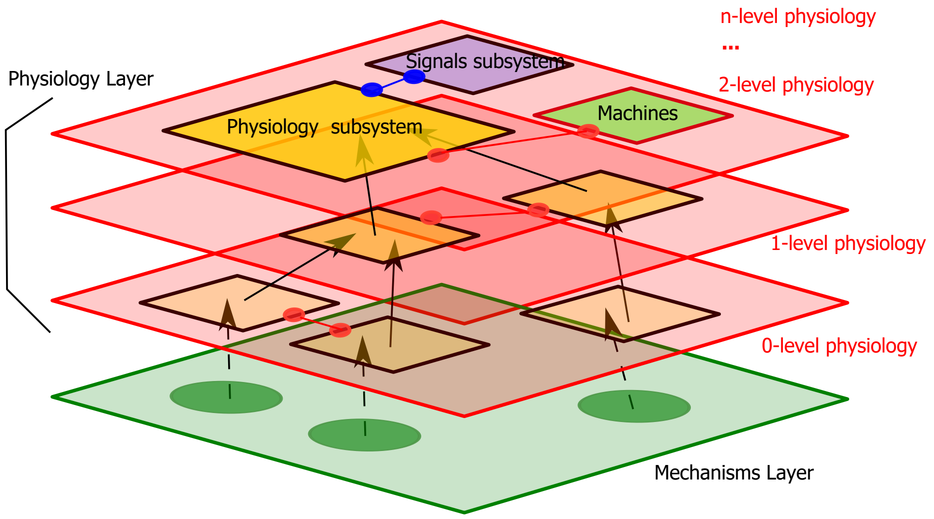


Figure 2. Conceptual structure of a CybSim model that shows the difference between multilevel (multiscale) and multilayer features for physiology entities and their mechanisms (dynamics). Machines and Signals subsystems are presented in the top level of the physiology layer to simplify the structure.

rated mechanistic layer. The fast evolution of the knowledge discovery in the biology field is the reason for this design criterion.

- Genes and Systems Biology. CybSim include metabolic networks based on a Systems Biology approach. This type of metabolic network is defined in the Mechanisms layer using a set of network components compliant with the Systems Biology Graphical Notation (SBGN), which requires the third enumeration type, `genes`. It is declared initially as the genesDummy type (see Listing 1), which must be redeclared in the final model if Systems Biology Metabolic Networks are used.

- Machines layer. A full machine-physiological system model is considered a cyborg model in CybSim. This generalization of a machine as a cybernetic component is based on the fact that near all machines designed for therapy, support or function enhancement include some type of automatic control systems.

- Signals layer. Data-driven or functional models (Roa and Prado 2006) are designed as Modelica blocks and organized in a package. PD standard models or PK metrics (e.g. Area Under the Curve, AUC) are included here.

A detailed description of the implementation of these features exceed the scope of this paper. However, some relevant issues are clarified in the following paragraphs.

Listing 1. Chemicals definitions and main CybSim scopes

```
type chemicalsDummy = enumeration(
 dummyCmp
 ) "Dummy Chemicals";
type genesDummy = enumeration(
 dummyGen
 ) "Dummy Genes";
// ...
/* Mechanisms configurations */
type PhysScopeType = enumeration (
 PBPK "Physiological -based
     Pharmacokinetics with parametric body
      Temperature",
 LPhys "Lumped Physiology",
 DPhys "Distributed Physiology"
 ) "Types of physiological approximations"
     ;
type chemicalVolumeType = enumeration (
 NoVolume "First one is the most simple
     mode",
 AllAsSolvent  "All compounds with the
     same specific volume that solvent",
 SmallAsSolvent "Non-small chemicals
     consider different volume"
 ) "Volumes of chemicals compared with
     solvent";
```

The physiological connectors depend on the scope type as expected, since blood flow rates are user defined with the PBPK scope, whereas they are solved according cardiovascular and circulatory system with the LPhys scope.

The Listing 2 shows the definition of a PBPK input connector as a causal connector that defines chemical and solvent flow rates, and the definition of a LPhys blood connector as an acausal connector that include chemical concentrations as Stream variables to address the reversion of blood flow rates. The later occurs for example in the arterio-venous fistula that connects a dialyzer with a patient. As a consequence, a PBPK model connected to a dialyzer cannot describe some operating conditions that occur with a clotted fistula in some patients. This is not a limitation of CybSim, but of the PBPK modeling approach.

The class bloodConnDummy allows controlling the types of Physiological connectors that may be redeclared in some multiscope CybSim components using the `constrainedBy` Modelica keyword.

Listing 2. Blood connectors

```
within CybSim;
encapsulated package Interfaces
 // ...
 partial connector bloodConnDummy
 end bloodConnDummy;
 partial connector inputDrag
  replaceable type chemicals =
      Properties.chemicalsDummy "Chemicals
      in Cyborg";
  input SI.VolumeFlowRate bvf (displayUnit
      = "l/min") "Causal input volume flow
      rate";
  input SI.MassConcentration c[chemicals](
      each displayUnit = "ug/l") "
      Concentration of chemicals";
 end inputDrag;
 // ...
 /*  PBPK connectors */
 connector inputBlood  "Blood input
     connector with PBPK scope"
  extends bloodConnDummy;
  extends inputDrag;
 end inputBlood;
 // ...
 /*  LPhys connectors */
 connector bloodPort
  extends bloodConnDummy;
   replaceable type chemicals =
       Properties.chemicalsDummy "Chemicals
        in Cyborg";
   SI.Pressure ps "static pressure at
       connection point";
   flow SI.VolumeFlowRate vf "inlet (>0)
       volume flow rate at connection port
       ";
   stream SI.MassConcentration c[chemicals
       ](each displayUnit = "ug/l") "
       Concentration of chemicals";
  end bloodPort;
```

A key declaration in Listing 2 is the `replaceable type chemicals` that is defined by a shorthand inheritance from the `chemicalsDummy` enumerated declared in the Listing 1. The type chemicals must be defined in the user's PBPK model according to the chemicals compounds required by the modeler. This feature is not available in other acausal languages as EcosimPro language (EL) (Empresarios Agrupados 2019).

A machine component may connected to any physiological subsystem, and therefore it must adapt their connectors and behaviour to the selected physiological scope. This is achieved thanks to redeclare the machine model and connectors as a function of the physiological scope type, during model building. The Listing 3 presents the technique used for the pharmaceutical drug form machine shown in Figure 3. In this case, the DrugConn connector and the DrugFormScope model that describes the drugForm behaviour may be redeclare, providing that connectors derive from bryConnDummy and machine models derive from drugFormDummy. The outer parameter PhysScope has the inner declaration in the final Cyborg model, where it is verified that selected connectors and machine models are compliant with the physiological scope. A deeper analysis exceeds the scope of the paper.

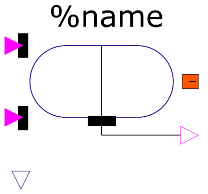


Figure 3. The drug form icon shows two boolean inputs (planned or instantaneous administration setting), one boolean output that informs if a significant drug amount remains to be dissolved, and a physical connector to the liberation lumen.

Listing 3. Main Modelica structure of the Ideal drug form machine

```
partial model drugFormDummy "Dummy base for
     drug pharmaceutical form"
end drugFormDummy;
model drugFormPBPK "Drug pharmaceutical
    form with PBPK scope"
 extends drugFormDummy;
 replaceable type chemicals =
     Properties.chemicalsDummy "Chemicals
     in Cyborg";
 outer parameter Properties.PhysScopeType
     PhysScope "Physiological scope of
     machine component";
 //...
end drugFormPBPK;
model drugFormLPhys "Drug pharmaceutical
    form with LPhys scope"
 extends drugFormDummy;
 replaceable type chemicals =
     Properties.chemicalsDummy "Chemicals
     in Cyborg";
 outer parameter Properties.PhysScopeType
     PhysScope "Physiological scope of
     machine component";
 // ...
end drugFormLPhys;
model drugForm "Drug pharmaceutical form"
 extends Icons.PharmaForm;
 extends innerParamsdrugForm;
 replaceable connector DrugConn =
     Interfaces.volPortPBPK constrainedby
     bryConnDummy
 DrugConn pDrug(redeclare type chemicals =
     chemicals) "Drug port"
 replaceable class drugFormScope =
     drugFormPBPK constrainedby
     drugFormDummy
 drugFormScope machine(redeclare type
     chemicals = chemicals);
 BooleanOutput bo "Boolean activation
     signal when there is drug mass"
 BooleanInput biPlan "Planned mode
     administration signal activation"
 BooleanInput biInst "Instantaneous mode
     administration signal activation"
```

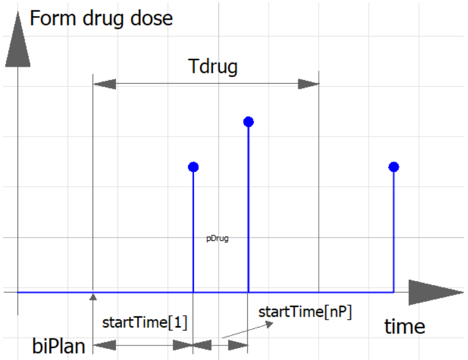


Figure 4. Main planned administration parameters in the drug form. Tdrug is the temporal period, startTime is the starting time of each drug administration in Tdrug, and the number of administrations is nP.

```
 equation
  connect(machine.pDrug, pDrug);
  connect(machine.bo, bo);
  connect(machine.biPlan, biPlan);
  connect(machine.biInst, biInst);
end drugForm;
```

The Drug form may perform both a planned and instantaneous pill administration, controlled by the boolean inputs `biPlan` and `biInst`. The Figure 4 shows the simplified diagram that appears in the drug form canvas to clarify the meaning of several planned mode parameters. The temporal area of the pill's surface is modelled according to its geometry. The area is considered by the connected liberation locus (physiology) as a function of the underlying physicochemical mechanism (Berrozpe, Lanao, and Guitart 2013, Ch. 19).

Other features of the CybSim design includes the execution of physiological algorithms to calculate partition ratios, in-vitro in-vivo extrapolation, and different types of physiological scaling as functions of the `BioData` package.

3.2 Mechanisms - Physiological layers

This Section addresses the key aspects of the CybSim design that supports the mechanisms - physiological multilayer feature. I have selected a flow limited tissue (FLT) (Berrozpe, Lanao, and Guitart 2013, Ch. 13) under the PBPK modeling approach (CybSim physiological scope) as a basic tissue that facilitates the description. In this respect, the aim of the study is not to present the full equations and assumptions of a full PBPK either LPhys model.

The Figure 5 describes the Modelica classes associated with the definition of the cited FLT component, called `sgnBryFlt`, which is shown in Figure 6. Besides the arterial and venous blood paths that connect to the tissue spatial region, the chemical compounds may be transferred in this tissue through a physiological barrier connected to the boundary port. The main variables related to the FLT dynamics may connected via the signal connections to data-driven models, as describen in the previous Section.

The FLT classes structure presented in Figure 5 was designed to allow the selection of the underlying physical mechanisms during model building. The eligible mechanisms for a PBPK component are organized according to functional types in the `sgnGeneralPBPK` partial component (Figure 5). A similar partial component, called `generalPBPK` is used if the physiological component does not require connectors to the Signals layer.

In agreement with the Figure 5, the fundamental structure of a PBPK FLT component appears in Listing 4. In this example, the mechanisms are inherited through the partial class `generalPBPK`. The equations section that appears commented in Listing 4 include very basic equations that complete the definition of the `flt` connector variables.

Listing 4. Flow limited tissue code structure

```
model flt "flow limited tissue"
 extends Icons.CausalEntity;
 extends VFP.generalPBPK;
 parameter Integer nBin = 1 "number input
     blood perfusion volume flow rate"
  annotation(Dialog(connectorSizing = true)
      , Evaluate = true);
 parameter Integer nBout = 1 "number of
     output blood perfusion volume flow
     rate vias"
    annotation(Dialog(connectorSizing =
        true), Evaluate = true);
 Interfaces.inputBlood[nBin] inVBlood(
     redeclare type chemicals = chemicals)
     "Input blood perfusions"
 Interfaces.outputBlood[nBout] outVBlood(
     redeclare type chemicals = chemicals)
     "Output blood perfusions"
equation
 /* Basic Equations related to systemic
     behaviour - connections */
 // ...
end flt;
```

As pointed in Figure 5, the physiological dynamics defined by the partial class `sgnGeneralPBPK` (and `generalPBPK`) is obtained through the inheritance of mechanisms organized by functional types. The code structure of `generalPBPK` partial class is shown in Listing 5. In opposition to `sgnGeneralPBPK` that inherits `volRegionSgnPBPK`, the component `sgnGeneralPBPK` inherits directly `innerWholeVarsPBPK` since it has not conditional interfaces to the Signals layer (see right column of Figure 5). A deeper description of this component exceeds the scope of the paper.

Listing 5. general PBPK dynamics of a volumetric region

```
partial model generalPBPK "General whole
    mechanism for volumetric regions with
    PBPK scope"
 extends Interfaces.innerWholeVarsPBPK;
 extends PP.MassBalance.MasterPBPK;
 extends PP.Elimination.MasterPBPK;
 extends PP.PhaseDistribution.MasterPBPK;
 extends PP.ChemicalActivity.MasterPBPK;
 extends PP.MolecularBinding.MasterPBPK;
 equation   // Implicit relationship among
     variables
```

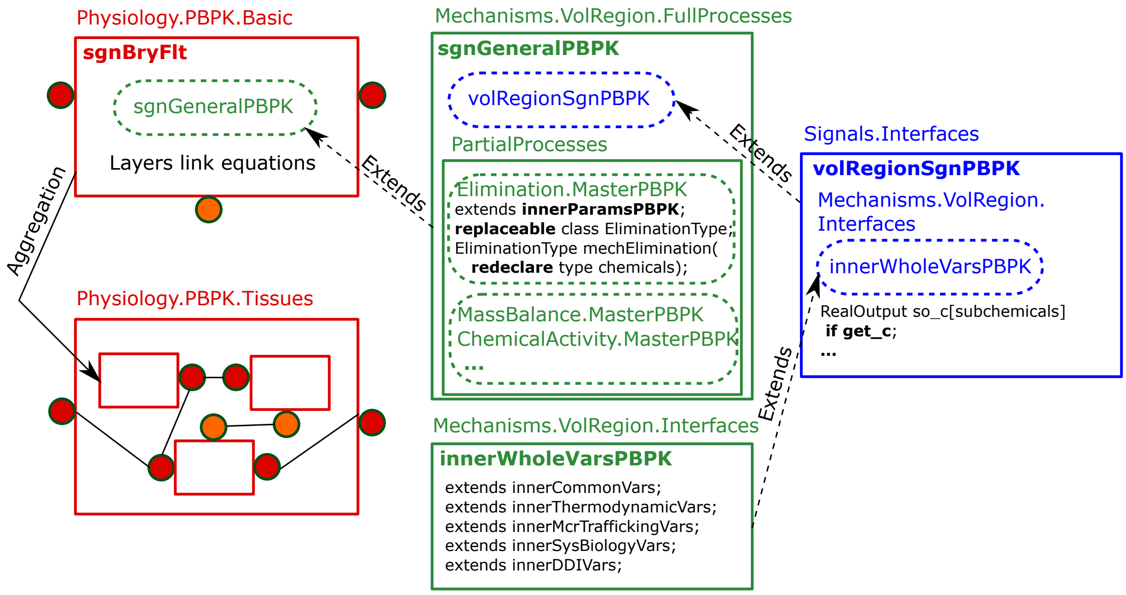


Figure 5. Simplified classes structure of the `sgnBryFlt` component (left blocks) in the Physiology layer, which inherits the mechanisms `sgnGeneralPBPK` where it is defined the component dynamics. The `sgnGeneralPBPK` partial component is defined through a set of replaceable classes. Each one describes a type of eligible behaviour (metabolism, elimination, binding, etc.) related to a spatial volumetric region (partial processes, middle blocks). They are defined through equations that govern the variables of the spatial region, declared as inner variables in the `innerWholeVarsPBPK` partial component. Partial mechanisms work with the associated outer variables. The `sgnGeneralPBPK` full mechanism inherits `volRegionSgnPBPK` (right block) that in turn inherits `innerWholeVarsPBPK` and declare some conditional structures that simplifies the connection of metrics and other functional models of the Signals layer to the `sgnBryFlt` component.

```
  // Concentrations
  for i in chemicals loop
   c[i]*vbulk = m[i];
  end for;
  // vbulk - mbulk and others
  msolvent = mbulk - sum(m[i] for i in
      chemicals);
  vsolvent = msolvent*solventPropT();
  if (chemVol ==
      Properties.chemicalVolumeType.NoVolume
      ) then
   vbulk = vsolvent;
  elseif (chemVol ==
      Properties.chemicalVolumeType.
      AllAsSolvent) then
   vbulk = mbulk*solventPropT();
  else  // SmallAsSolvent
   ...
  end if;
  // input bulk mass flow rate
  mfbulkin = mfbulkinDuct + mfbulkinMem;
  // Generation g and group - related
  gbulk = gsolvent + sum(g);
end generalPBPK;
```

The eligible physical mechanism (see Figure 5) is called `MasterPBPK` and it is organized in packages according to the behaviour type. The Listing 6 shows the code of `MasterPBPK` for the elimination behaviour. The class `dummyPBPK` is defined to control the eligible elimination mechanisms.

Listing 6. Master model of the elimination dynamics in a volumetric region

```
partial model MasterPBPK "Master model for
    Elimination mechanisms with PBPK scope"
  extends innerParamsPBPK;
  replaceable class EliminationType =
      NullPBPK constrainedby dummyPBPK
   annotation(choicesAllMatching = true);
   EliminationType mechElimination(
     redeclare type chemicals = chemicals,
     redeclare type mcrChemicals =
       mcrChemicals,
     redeclare type genes = genes
     );
end MasterPBPK;
```

The default elimination mechanism, `Elimination.NullPBPK` is the null elimination mechanism. Any volumetric region with this elimination mechanisms does not eliminate chemical compounds. The linear plus Michaelis - Menten (saturable) elimination is a well-known mechanism that describes this type of behaviour in Pharmacokinetics. The removal of any chemical i according this one is as follows:

$$e_i = (K_{e,i} + \frac{V_{em,i}}{K_{em,i} + C_i}) \cdot C_i \,, \qquad (1)$$

in which the concentration C_i is equal to the unbound significant phase (tissue) concentration $c_{u,i}$ if the mechanism's parameter `significantPhase` is true, and to the unbound non-significant phase (venous) concentration, $c_{nu,i}$ otherwise. In homogeneous volumetric regions both concentrations are the same.

Assuming that `significantPhase` is true, the Equation 1 may also be written as follows:

$$e_i = Cl_i \cdot c_{u,i} \,, \qquad (2)$$

in which the term Cl_i is the intrinsic clearance of chemical i in the region and phase considered, given as:

$$Cl_i = K_{e,i} + \frac{V_{em,i}}{K_{em,i} + c_{u,i}} \tag{3}$$

The variables $c_{u,i}$ must be substituted by $c_{nu,i}$ in equations (2) and (3) if the elimination occurs in the non-significant phase. A detailed description of the tissue intrinsic clearance for a well-stirred region is shown in (Pang and Malcolm Rowland 1977). The elimination mechanism related to Equation 2 is implemented in Listing 7.

Listing 7. Saturable elimination dynamics in volumetric region

```
partial model clearanceSat  "Sat+Lin
    elimination in volumetric region"
  extends outerCommonVars;
  outer parameter Boolean significantPhase
      "Place of elimination phase for
      chemicals";
  outer parameter SI.VolumeFlowRate Ke[
      chemicals] "Clearance of chemicals";
  outer parameter SI.VolumeFlowRate Kesolv "
      Clearance of solvent";
  outer parameter SI.MassConcentration  Kem[
      chemicals] "Michaelis Menten constant"
      ;
  outer parameter SI.MassFlowRate Vem[
      chemicals] "Michaelis Menten velocity"
      ;
equation
  if significantPhase then
   for i in chemicals loop
    e[i] = cu[i]*(Ke[i] + Vem[i]/(Kem[i] +
        cu[i]));
   end for;
  else
   for i in chemicals loop
    e[i] = cnu[i]*(Ke[i] + Vem[i]/(Kem[i] +
        cnu[i]));
   end for;
  end if;
  //...
end clearanceSat;
//
partial model clearanceSatPBPK "Sat+Lin
    elimination in volumetric region with
    PBPK scope"
  extends dummyPBPK;
  extends clearanceSat;
end clearanceSatPBPK;
```

The variables that define the dynamics in a volumetric region are defined in the partial class `outerCommonVars`, which is accesible to any single mechanism. The associated inner parameters are defined in the Master component, inherited from `innerParamsPBPK` (Listing 6 and Figure 5). A deeper description of these outer-inner definitions exceeds the scope of the paper.

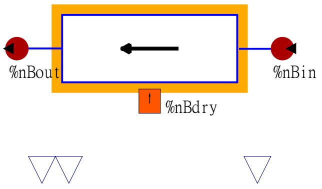


Figure 6. Flow limited tissue (flt) PBPK component with blood ports (circles left-right), boundary port (rectangle below) and signal connections (triangles) `sgnBryFlt`.

The mechanism that defines the elimination behaviour is selected during the procedure of building the model, through the declaration of the `sgnBryflt` physiological component, as seen in the Listing 8 for the `central` instance. As seen, the value of the $K_{e,i}$ linear parameter is modified from its default value, using a 3-vectorial expression, what indicates that three chemical compounds are defined.

Listing 8. Declaration of a FLT instance (central) in a PBPK model

```
CybSim.Physiology.PBPK.Basic.sgnBryFlt
    central(
  redeclare type chemicals = chemicals,
  redeclare class EliminationType =
      CybSim.Mechanisms.VolRegion.
      PartialProcesses.Elimination.
      clearanceSatPBPK, Ke = {0, 5.5e-6, 8.3
      e-6})
```

4 Study case

Figure 7 shows the diagram of the model defined in the Methods Section. A detailed description of this one appears in (Mangas-Sanjuan et al. 2018), whereas the comparative analysis of the PhysPK vs NONMEM implementations may be seen in (Reig-Lopez et al. 2020). The building of this model was performed using the GUI of openModelica 1.14.1 (diagram view), although the selection of non-default mechanisms for the physiological components were completed in the Modelica code view of the model, because openModelica 1.14 does not include this graphical function.

The model has been parameterized for a low dose of PD (100 mg) administered with two intakes of 50 mg separated 12 h, for a high absorption rate constant in the gut (drug of class II of Biopharmaceutics Classification System), saturable metabolism in gut and liver, and a reference value of the dissolution rate constant (quality level).

The parameter values agree with those used in the first case of (Mangas-Sanjuan et al. 2018, Fig. 2), excepting the division of the 100 mg PD dose in two separated 50 mg PD doses, which is applied now in agreement with the second step pointed in Methods to analyze a more complex scenario, after validating the accuracy of the model against the results of (Mangas-Sanjuan et al. 2018) and (Reig-Lopez et al. 2020).

The final objective of this case study is to demonstrate the feasibility of CybSim to select and change the underlying mechanisms of the model physiological instances during the building process. With this goal, after simulating the model with the liver mechanisms defined in (Mangas-Sanjuan et al. 2018) (first simulation), a linear molecular binding mechanism is added to the liver and a new simulation is executed (second simulation).

The Listing 9 shows the structure of the Modelica code of the semiphysiological model, including the definition of the Liver tissue. The `chemicals` enumeration is set through the three required chemicals compounds, PD, PM, and SM. The mechanism associated with the chemical activity in the liver is redeclared as a saturable metabolism (`MichaelisMentenPBPK`), during the initial model building. The boolean inputs `biInst`

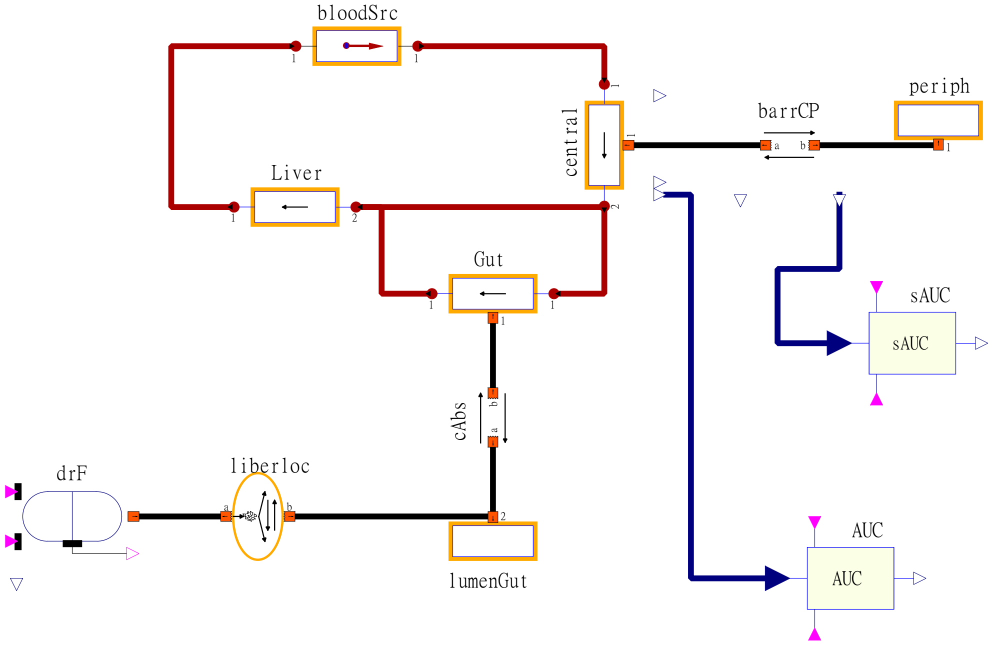


Figure 7. Semiphysiological PBPK model published in (Mangas-Sanjuan et al. 2018; Reig-Lopez et al. 2020) that includes a drug form (drF) from the machine layer, and two PK metric blocks (AUC and sAUC) from the signals layer.

and `biPlan` define a true planned and false instantaneous administration mode for the drug form, as wished.

The execution of the model for a temporal window of 24 hours gives the plasmatic concentrations of PD, PM and SM under the defined scenario (first simulation).

Listing 9. Main code structure and liver definition of semiphysiological model

```
model drF2cmpLivGut
  extends
      CybSim.ModelsInterfaces.Interfaces.
      pbpkStr(redeclare type chemicals =
      chemicals);
  import CybSim.Properties;
  import PQ =
      CybSim.BioData.PhysicalChemical;
  /* Chemical definitions */
  type chemicals = enumeration(PD "Parent
      drug", PM "Primary metabolite", SM "
      Secondary metabolite");
  CybSim.Physiology.PBPK.Basic.flt Liver(
    redeclare type chemicals = chemicals,
    redeclare class ChemicalActivityType =
        CybSim.Mechanisms.VolRegion.
        PartialProcesses.ChemicalActivity.
        MichaelisMentenPBPK,
    Kmm = {0.1, 0.01, 0.01}, Vmm = {0, 1e-9,
        1e-9}, reactantsm={{
        chemicals.PD,chemicals.PD,chemicals.PD
        }},
    Vtis0 = 0.003, nBin = 2, nBout = 1)
  //...
equation
  // ...
  drF.biInst = false;
  drF.biPlan = true;
end drF2cmpLivGut;
```

The Listing 10 shows how a linear molecular binding is applied to substitute the default (null) molecular binding of the liver instance of the semiphysiological model.

The linear molecular binding mechanism modifies the amount of free drug according to the unbound fraction drug values $f_{u,i}$. These fractions are defined in the liver declaration as 5% (0.05) for the PD, and 100% (1) for the metabolites. That is, the PD is the unique compound that is bound to a macromolecule, in such a way that only 5% is free. This is a very common situation in physiological models.

Listing 10. Definition of the FLT liver instance with linear binding

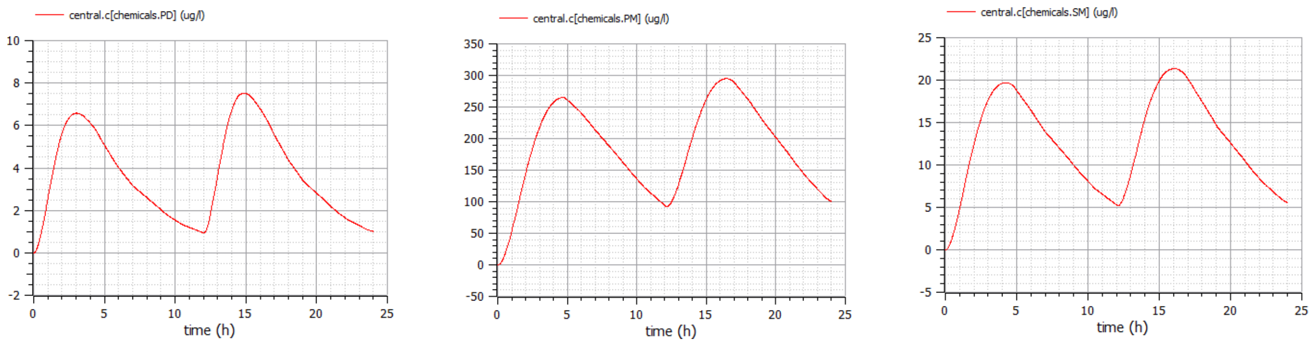
```
CybSim.Physiology.PBPK.Basic.flt Liver(
    redeclare type chemicals = chemicals,
  // ...
  redeclare class MolecularBindingType =
      CybSim. Mechanisms.VolRegion.
      PartialProcesses.MolecularBinding.
      linearPBPK, fu = {0.05, 1, 1}, ...)
```

The execution of the model for the same temporal window of 24 hours gives the plasmatic concentrations of PD, PM and SM in the second simulation.

5 Simulation results and discussion

Figure 8 shows the plasmatic (central) concentrations of the parent drug and their metabolites during the first 24 hours, starting with the first PD dose (50 mg).

The temporal distribution of the chemical compound was accurately validated both for an unique dose of PD equal to 100 mg (Mangas-Sanjuan et al. 2018, Fig. 2) and for two sequential doses of PD equal to 50 mg. A detailed analysis of this testing phase exceeds the scope of this paper that is focused to demonstrate the feasibility of the CybSim design to support the mechanisms - physiology architecture.

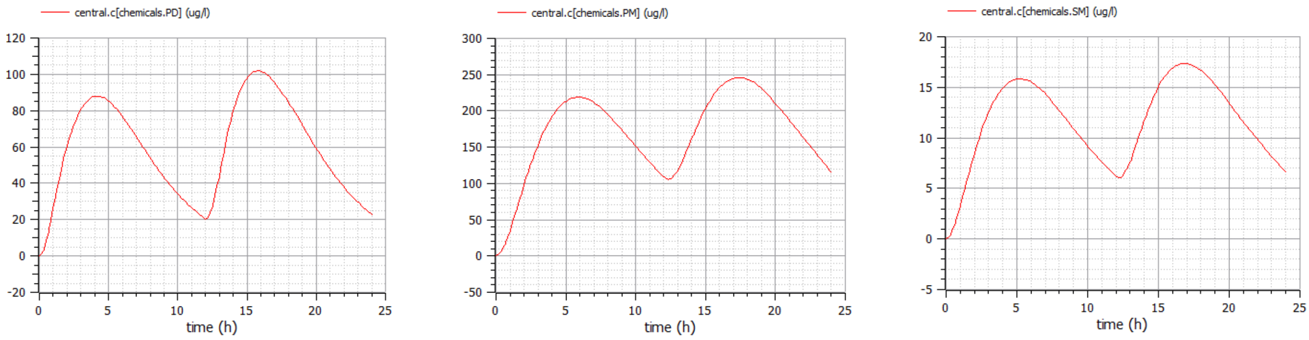


(a) Parent Drug.

(b) Principal metabolite.

(c) Secondary metabolite.

Figure 8. Plasma (central) concentrations for a low dose scheme (50+50 mg) without liver molecular binding.



(a) Parent Drug.

(b) Principal metabolite.

(c) Secondary metabolite.

Figure 9. Plasma (central) concentrations for a low dose scheme (50+50 mg) with liver molecular binding for PD ($f_u = 0.05$).

The Figure 9 shows the plasmatic concentrations of PD and their metabolites during the first 24 hours for the second simulation. As expected, the PD concentration increases, whereas PM and SM concentrations decrease, with respect to the first simulation, due to the reduction of liver metabolism because of the smaller amount of free PD.

The Area Under the Curve (AUC) blocks, which are connected to the concentrations (all the chemicals) inside the central compartment (AUC block) and to the total mass flow rate from central to peripheral compartment (sAUC or single AUC), are not presented here. However, they were used to evaluate the parent drug and metabolites AUC in the target central compartment and to calculate the total net amount of mass between central and peripheral compartment during the first 24 horas. They were used also to demonstrate the capabilities of CybSim related to the Signals layer.

Although a detailed comparison between CybSim and PhysPK exceeds the scope of this paper, CybSim overcomes several important limitations of PhysPK due to the lack of classes' redeclaration, inner/outer structures, packages organization, and causality of connectors. These language characteristics are the basis of the mechanisms - physiology multilayer, the multiscope feature, the free definition of chemical compounds inside the physiological instances, and many of the capabilities of the signals layer.

Summary

This study has presented a novel Modeling and Simulation software system for the biosciences field, CybSim, that gives a framework compliant with current methodologies and specific solutions required in particular areas like pharmacology, precision medicine and toxicology. CybSim tries to overpass some detected lacks related to model reusing and sharing in current Modeling and Simulation software systems.

The outcomes demonstrate the feasibility of CybSim to facilitate the choice of the mechanisms underlying the physiological entities in the process of model building. To the best of my knowledge, this is the first biosimulation system that fulfils that feature at the same time that offers modeling multiscope, free definition of chemicals, multilevel modeling, metabolic networks based on the systems biology approach, machines integration and support for data-drive modeling.

Future works will be developed to evaluate those features, and to complete the implementation of other packages related to optimization (applied to population estimation and dosage personalization), and biodata (algorithms for in-vitro in-vivo correlation, allometric scaling, and methods for computation of physiological properties).

This is a first paper concerning the preliminary version 0.2 of the CybSim biosimulation system. More advances will be performed and published shortly, including the evolution to the available openModelica 1.18 that should give better solutions to some relevant planned features. CybSim will be deployed with availability for download under open source license, after reaching the required minimal functionality.

References

Azer, Karim et al. (2021). "History and Future Perspectives on the Discipline of Quantitative Systems Pharmacology Modeling and Its Applications". In: *Frontiers in Physiology* 12.

Berrozpe, José Doménech, José Martinez Lanao, and Concepción Peraire Guitart (2013). *Tratado general de Biofarmacia y Farmacocinética. Volumen II [General treatise of biopharmacy and pharmacodynamics. Vol II]*. Madrid, Spain: Editorial Síntesis.

Bizzotto, R. et al. (2017). "PharmML in Action: an Interoperable Language for Modeling and Simulation". In: *CPT Pharmacometrics Syst Pharmacol* 6.10, pp. 651–665.

Bloomingdale, P. et al. (2017). "Quantitative systems toxicology". In: *Curr Opin Toxicol* 4, pp. 79–87.

Bonate, Peter L. (2011). *Pharmacokinetic-Pharmacodynamic Modeling and Simulation*. Second Edition. Springer, p. 618. ISBN: 9781441994844.

Darwich, A. S. et al. (2017). "Why has model-informed precision dosing not yet become common clinical reality? lessons from the past and a roadmap for the future". In: *Clin Pharmacol Ther* 101.5, pp. 646–656.

Empresarios Agrupados (2019). *User Manual EcosimPro 6.0*. Report.

Fritzson, Peter (2015). *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3*. second edition. USA: Wiley, p. 1250. ISBN: 9781-118-859124.

GastroPlus (2021). *GastroPlus. Available online: www.simulations-plus.com/software/gastroplus (accessed on 10 May 2021)*. Web Page.

Gonzalez-Garcia, Ignacio et al. (2017). "Comparison of FO – FOCE population parameter estimation methods in PhysPK 2.0 against NONMEM 7.3". In: *PAGE 2017. Abstracts of the Annual Meeting of the Population Approach Group in Europe*, pp. 1–2.

Hester, R. L. et al. (2011). "HumMod: A Modeling Environment for the Simulation of Integrative Human Physiology". In: *Front Physiol* 2, p. 12.

Hunter, Peter J. (2006). "Modeling Human Physiology: The IUPS/EMBS Physiome Project". In: *Proceedings of the IEEE* 94, pp. 678–691.

Hunter, Peter J. and Marco Viceconti (2009). "The VPH-Physiome Project: Standards and Tools for Multiscale Modeling in Clinical Applications". In: *IEEE Reviews in Biomedical Engineering* 2, pp. 40–53.

Ježek, Filip et al. (2017). "Lumped models of the cardiovascular system of various complexity". In: *Biocybernetics and Biomedical Engineering* 37.4, pp. 666–678.

Jones, Hannah M, Iain B Gardner, and Kenny J Watson (2009). "Modelling and PBPK simulation in drug discovery." In: *The AAPS journal* 11, pp. 155–166.

Keating, S. M. et al. (2020). "SBML Level 3: an extensible format for the exchange and reuse of biological models". In: *Mol Syst Biol* 16.8, e9110.

Maharao, N. et al. (2020). "Entering the era of computationally driven drug development". In: *Drug Metab Rev* 52.2, pp. 283–298.

Mangas-Sanjuan, V. et al. (2018). "Computer simulations for bioequivalence trials: Selection of analyte in BCS class II and IV drugs with first-pass metabolism, two metabolic pathways and intestinal efflux transporter". In: *Eur J Pharm Sci* 117, pp. 193–203.

Mateják, M. et al. (2014). "Physiolibrary – Modelica library for physiology". In: *10th International Modelica conference*.

Modelica Association (2017-04). *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.

NONMEM (2021). *NONMEM. Available online: www.iconplc.com/innovation/nonmem/ (accessed on 17 Jul 2021)*. Web Page.

Open-Systems-Pharmacology (2021). *Open-Systems-Pharmacology. Available online: www.open-systems-pharmacology.org (accessed on 10 May 2021)*. Web Page.

Paini, A. et al. (2019). "Next generation physiologically based kinetic (NG-PBK) models in support of regulatory decision making". In: *Comput Toxicol* 9, pp. 61–72.

Pang, K. Sandy and Malcolm Rowland (1977). "Hepatic Clearance of Drugs. I. Theoretical Considerations of a "Well-Stirred" Model and a "Parallel Tube" Model. Influence of Hepatic Blood Flow, Plasma and Blood Cell Binding, and the Hepatocellular Enzymatic Activity on Hepatic Drug Clearance". In: *Journal of Pharmacokinetics and Biopharmaceutics* 5.6.

Polasek, T. M., S. Shakib, and A. Rostami-Hodjegan (2019). "Precision medicine technology hype or reality? The example of computer-guided dosing". In: *F1000Res* 8, p. 1709.

Prado-Velasco, Manuel (2016). "Bridging the gap between open and specialized modelling tools in PBPK/PK/PD with PhysPK/EcosimPro modelling system: PBPK model of methotrexate and 6-mercaptopurine in humans with focus in reusability and multilevel modelling features". In: *PAGE. Abstracts of the Annual Meeting of the Population Approach Group in Europe*, pp. 1–2.

Prado-Velasco, Manuel, Alberto Borobia, and Antonio Carcas-Sansuan (2020). "Predictive engines based on pharmacokinetics modelling for tacrolimus personalized dosage in paediatric renal transplant patients". In: *Scientific Reports* 10.1, p. 7542.

Reig-Lopez, J. et al. (2020). "A Multilevel Object-Oriented Modelling Methodology for Physiologically-Based Pharmacokinetics (PBPK): Evaluation with a Semi-Mechanistic Pharmacokinetic Model". In: *Computer Methods and Programs in Biomedicine* 189, pp. 1–11.

Roa, Laura and Manuel Prado (2006). "Simulation Languages". In: *Wiley Encyclopedia of Biomedical Engineering*. Ed. by Metin Akay. John Wiley and Sons, Inc., pp. 3186–3198. ISBN: 978-0-471-24967-2.

Rowland, M, Lj Lesko, and A Rostami-Hodjegan (2015). "Physiologically Based Pharmacokinetics Is Impacting Drug Development and Regulatory Decision Making". In: *CPT: Pharmacometrics & Systems Pharmacology* 4, pp. 313–315.

Sauro, Herbert M. et al. (2004). "Next Generation Simulation Tools: The Systems Biology Workbench and BioSPICE Integration". In: *OMICS: A Journal of Integrative Biology* 7.4.

Shepard, T et al. (2015). "Physiologically Based Models in Regulatory Submissions : Output From the ABPI / MHRA Forum on Physiologically Based Modeling and Simulation". In: pp. 1–5.

SimCyp (2021). *SimCyp. Available online: www.certara.com/software/simcyp-pbpk (accessed on 10 May 2021)*. Web Page.

Decarbonization of Industrial Energy Systems: A Case Study of Printed Circuit Board manufacturing

Carles Ribas Tugores[1] Gerald Birngruber[2] Jürgen Fluch[1] Angelika Swatek[3] Gerald Schweiger[2]

[1]AEE INTEC, Gleisdorf, Austria {c.ribastugores,j.fluch}@aee.at
[2]Technical University of Graz, Graz, Austria {birngruber,gerald.schweiger}@tugraz.at
[3]ENERTEC - Naftz & Partner GmbH & Co KG, Graz, Austria a.swatek@enertec.at

Abstract

Decarbonization of industry is a key challenge to achieve the Paris climate goals. Digitalization of the industry is a cornerstone of this journey. In this paper we present our modelling work towards the creation of a Digital Energy Twin of the energy supply system of a printed circuit board manufacturing by means of a classical use case, system design optimization. The simulation approach allowed us to fairly compare the improvements done in the energy supply system by evaluating those under the same operating conditions. Integration of chiller's waste heat can cover most of the low temperature grid heat demand while the additional generation of chilled water reduces the amount of water pump from and back to the river.

Keywords: Digitalization, Industry, Modelling, Efficiency, Decarbonization

1 Introduction

The industry sector is one of the largest energy consumers and greenhouse gas emissions contributors. The chemical sector is the biggest industrial energy consumer, accounting for 28 % of total global industry final energy demand (Philibert 2017). The industrial sector must reduce its energy intensity and dependency on fossil fuels demands to achieve the commitment of the Paris agreement to hold global warming below 1.5-2°C (Luderer et al. 2018). A new draft of Effort Sharing Regulation specifies a 36 % emission reduction for Austria by 2030 compared to 2005 for sectors not covered by the emissions trading system (European-Commission 2019). While the power sector shows significant reductions in the different future technology scenarios, the share of industrial CO2 emissions will increase to 44 % in the 2 °C scenario, and it has not yet attracted the same level of attention as the transport and power sectors (Philibert 2017). Three-quarters of the worldwide industrial energy demand is dedicated for process heat, with 52 % of that energy required in the low and medium temperature level (Luderer et al. 2018). This shows that the target of 80 % CO2 reduction can only be reached if the process heat demand in the low and medium temperature range is incorporated in the energy reduction strategies.

The scientific community argues that digitization offers opportunities for sustainability, such as improved resource efficiency through optimized operation (Ghobakhloo 2020). A key concept in the digitalization are Cyber-Physical Systems (CPS) and Digital Twins (DT) (J. Lee, Bagheri, and Kao 2015; Tao et al. 2019). The emergence of these concepts poses new challenges for traditional modeling approaches. Among other aspects, computational systems and communication networks need to be combined with physical systems (E. A. Lee and Seshia 2017); further, co-simulation approaches are needed to couple different tools and modeling approaches (e.g., physical and machine learning) (Schweiger, Engel, et al. 2018).

In this paper, we present our ongoing modeling work towards a Digital Energy Twin (DET) in the industry by means of a real world case studies of the energy supply system of printed circuit board manufacturing plant at AT&S. AT&S is a world leading company in the printed circuit board industry (PCB). The goal of AT&S is to reduce the carbon footprint and fresh water use by yearly 5 % and 3 % respectively (AT&S 2021). The optimization of the energy system is not an easy task as production plants need to be regularly adapted to the customer needs. AT&S is managing to improve the efficiency of their plants but at the same time facing difficulties evaluate the exact impact of the taken measures and potential deviation with the expected results, i.e. to achieve an "optimum". We aim to develop a Digital Energy Twin targeted to assist on the optimization of the operational control of the system. Due to the lack of a common understanding one the definition of the Digital Twin concept it is important to note that thee authors follow the definitions for Digital Model, Digital Shadow and Digital Twin given by (Kritzinger et al. 2018). In a brief way, given a physical object and its counterpart digital object, the three concepts differ on the level of data integration. In a Digital Model the data flow between physical and digital object occurs manually. On a Digital Shadow, the information flowing from the physical to the digital object is automated. Thus, a change on the physical object will be automatically communicated to the digital model. In the Digital Twin the data flowing from physical to a digital object and vice-versa are automated.

A reasonable implementation process of a Digital Twin starts with a Digital Model, which data flow is step by step automated. This is a resource intensive task which requires in-deep knowledge of the system as well as large amount of data. Due to the industry needs, it is clear that not only a Digital Twin (which planned use cases are rather related to the optimal operation of the system) is of interest but a Digital Model (suitable to overall control strategies and system design offline studies) as well. Thus, we intend to efficiently use the resources being use to obtain a Digital Twin by making use of its "simplest" version, i.e. the Digital Model. The presented use case, scenario evaluation, aims to evaluate the added value of the undertaken measures to improve the efficiency of the plant by means of three exemplary scenarios. The plant is based on the real system, though only a part of the whole system has been included here.

2 Use Cases

Figure 1 shows an overview of the three scenarios considered. A subset of the actual warm and chilled water production system at an industrial site located in Austria.

The cooling, heating, and water demand at AT&S can be clustered into five main consumers: Two heat consumers (a high and a low temperature grid). Data of their temperature and energy requirements are available at their main heat exchangers. HT grid supply temperature varies between 80 °C and 50 °C. LT grid supply temperature varies between 40 °C and 20 °C. The cooling demand is divided into two consumers, a main cooling demand with supply temperatures between 10 °C and 6.5 °C, and a cooling demand for industrial processes, production in short. The supply temperature for production should not be lower than 11 °C (the supply temperature is set to 12 °C). The fifth main consumer corresponds to the process water. It requires water at warm temperature level. Exact demands for the industrial process and process water are available as hourly average values.

Scenario 1 is the base case and corresponds to a former case where the use of the utilities was not optimized. The base case is depicted at Figure 1 with help of arrows with black edges. The heat demand is entirely supplied by a gas boiler. The cooling demand is solely covered by a compression chiller which waste heat is released to the ambient by a dry cooler. The cooling demand for the industrial process (production) is covered by water from a cold water storage. The mass flow leaves production with a slightly higher temperature and is send into the warm water storage. The warm water is later used to supply the process water demand (this water cannot be reused for the industrial process and is therefore treated and dumped safely). In case that the water level of the warm water storage is too high, water is pumped back into the river (warm water overflow). Water from the warm water storage is partially re-injected into the cold water storage to temperate the cold water storage (overcooling protection) and thus minimize the amount of water pumped back into the river. The amount of water in the system is refilled with fresh water to keep the cold water storage level above a minimum threshold.

Scenario 2 is a successor of scenario 1 and integrates the chillers' waste heat. The waste heat is used to partly supply the heat demand of the LT grid, see arrows with dashed edges in Figure 1, thus reducing the amount of heat needed to be supplied by the boiler plant and with it the overall gas consumption.

Scenario 3 is an extension of scenario 2. Scenario 3 aims to reduce the amount of water overflowing the warm water storage, and that needs to be pumped back into the river. Here the condenser side of the chiller is indirectly connected to the cold and warm water storage tanks by a heat exchanger. The chiller can increase the amount of chilled water produced, and this water is used to cool down fluid from the warm water storage, which is re-injected back to the cold water storage. The amount of fresh water needed is then reduced, as well as the amount of water overflowing in the warm water storage that needs to be pumped back into the river. Because the supply of warm water for the process water needs to be ensured, the generation of additional chilled water only takes place when there is enough warm water. A hysteresis block with uLow and uHigh equal to 30 % and 90 % of the water level ensures the warm water supply for the consumer "process water". The additional chilled water yields an increase on waste heat. The additional chilled water is only generated under suitable conditions, i.e. there is enough heat demand at the LT grid and the additional waste heat can be thus used to further cover these need.

3 Method

3.1 Modeling and Simulation

The models were implemented in the Modelica language (Fritzson and Engelson 1998). A discussion of limitations and promising approaches of the Modelica language can be found here (Schweiger, Nilsson, et al. 2020). The main reason to opt for Modelica is the compliance of most Modelica tools with the FMI standard. Notice that the developed modelica models for the energy supply system (partially here presented) as well as Python data driven models for selected production processes are later to be exported as FMUs and imported into a project partner software (KG 2021) where the Digital Twin is to be hosted. Another important aspect to decide for Modelica is the amount of already free-available libraries for energy systems and the acquaintance of the authors with those libraries. In this regard, the models are based on the Modelica IBPSA Project 1 (Wetter, Treeck, et al. 2019) and the Buildings Library (Wetter, Zuo, et al. 2014). Dymola was used to simulate Modelica models (Brück et al. 2002).

3.2 Model description and parametrization

A top level view of model is shown in Figure 2. The main subsystem models are here briefly described.

DOI
10.3384/ecp21181497

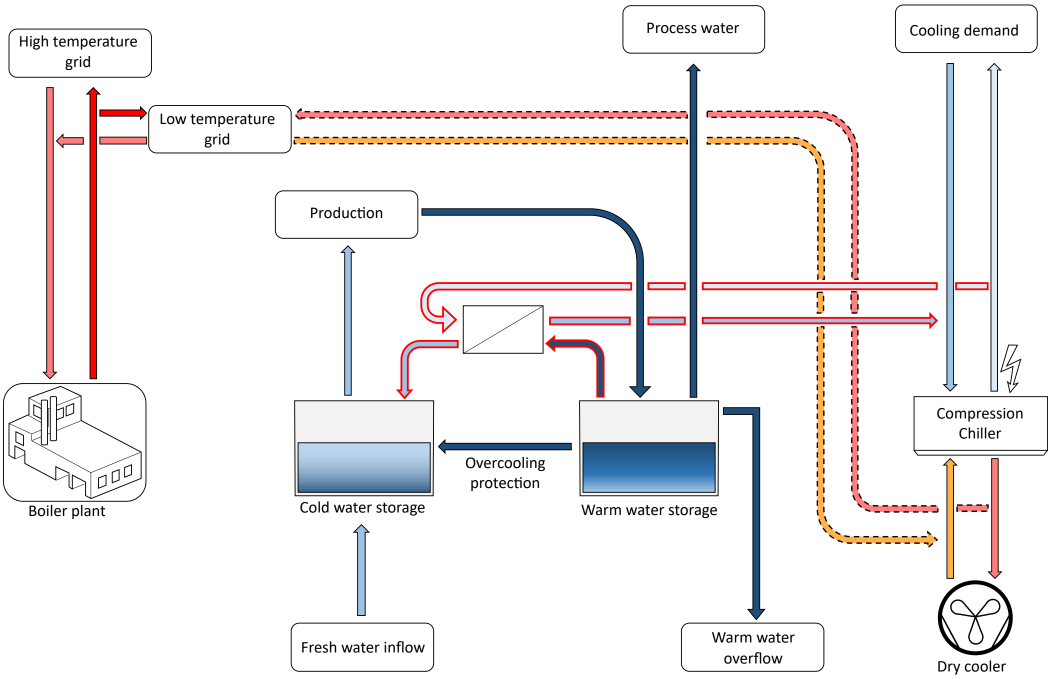


Figure 1. Schema of the industrial site. Arrows indicate mass flow rates. Arrows with dashed edges correspond to the scenario 2 and 3. Arrows with red edges correspond to scenario 3. Colours of the arrows indicate the temperature levels of the streams qualitatively.

The consumers "high temperature grid", "cooling demand", "production" and "process water" are modelled with a custom model named "GenericDemand". In this block the return temperature is prescribed. An instance of Buildings.Fluid.Interfaces.PrescribedOutlet ensures that the inflow mass flow rate is heated up (or cooled down) according to the measured data. The mass flow rate going to each consumer is regulated so that the energy demand is fulfilled. Notice that the mass flow rate leaving "process water" is flowing into a sink and not back to the water storages. The low temperature grid block contains a "GenericDemand" block inside. Here the main difference with the other demand blocks is that the mass flow rate used to cover the heat demand can come from two different sources, the boiler plant and/or the compression chiller. The waste heat coming from the compressor has priority, heat from the boiler plants is used in case there is a lack of waste heat or the waste heat supply temperature is not high enough.

The cold and warm water tanks are modelled using two instances of Modelica.Fluid.Vessels.OpenTank. These are parametrized based on constructive details of the real tanks. Total cross area and height are respected. Heat losses are not considered.

The "storage cooling HX" is modelled with a heat exchanger model with constant effectiveness, $\varepsilon = 0.8$. The mass flow rate on the chiller side is determined by the overall control as explained in section 2. The mass flow rate in the storage side is regulated so that the outflow temperature, i.e. mass flow rate flowing into the cold water, reaches a temperature of 13 °C.

The overcooling protection block contains mainly a pump moves fluid from the warm water storage to the cold one when the temperature of the cold storage drops below its minimum allowed of 11 °C.

The cold and water storage logic controls is based on models from the StateGraph library. It monitors the temperature and water levels of the warm and cold storage and includes the necessary logic to "activate/deactivate" different subsystems, e.g. the overcooling protection block, fresh water supply and overflow (to keep water level of the storages within certain limits) or the chiller (additional generation of chilled water used in scenario 3).

The boiler plant is modelled using an instance of Buildings.Fluid.Boilers.BoilerPolynomial. The nominal power of the boiler is set to 3.5 MW . The efficiency is defined by a polynomial which coefficients are obtained by curve fitting to measured data, see Figure 3.

The chiller installed at AT&S is TCHVBZ 31630 BT from the manufacturer Rhoss. It is modelled using the Buildings.Fluid.HeatPumps.ScrollWaterToWater model. The necessary information to parametrize the model cannot be directly obtained from a data sheet. For that purpose the buildings library supplies Python code. It mainly consists of the same implementation of the heat pump and refrigerant properties available in Modelica as well as a script. Given data on specific operating conditions (each defined by inflow temperature, mass flow rate and heat flow rate at the evaporator and condenser as well as the electrical consumption of the compressor) the script simulates the Python chiller model iteratively for all operating points adjusting the parameters

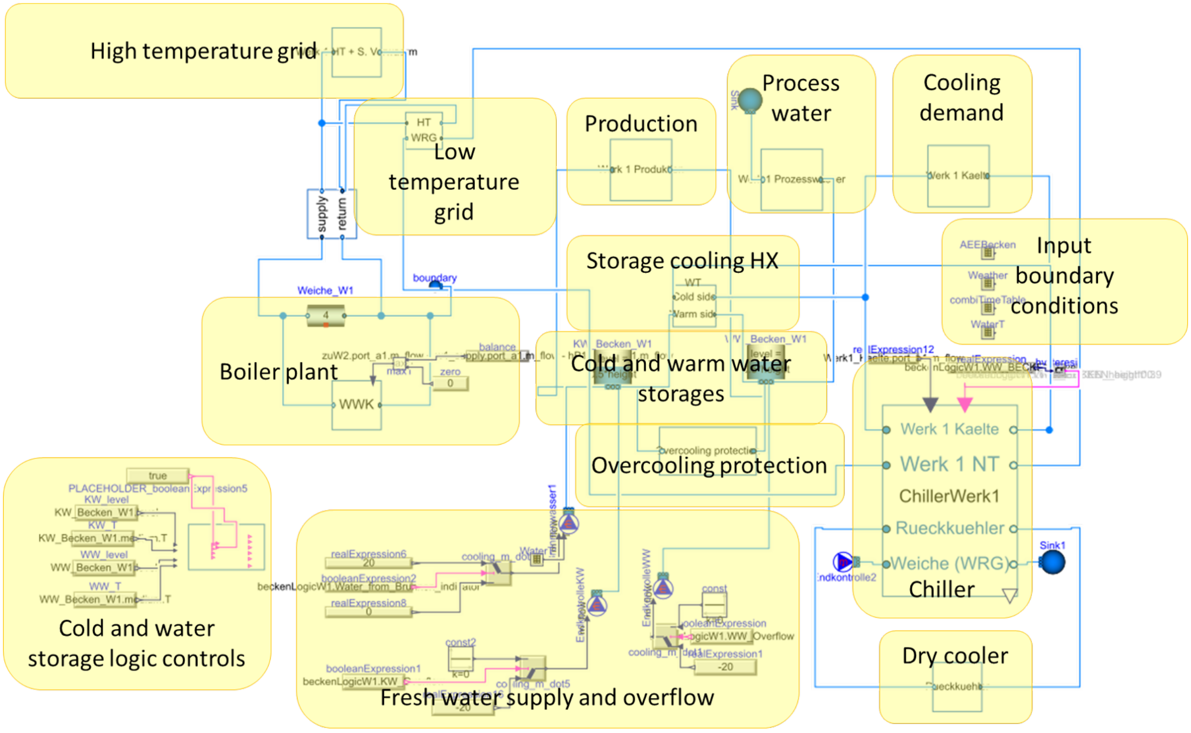


Figure 2. Top level view of the Modelica model.

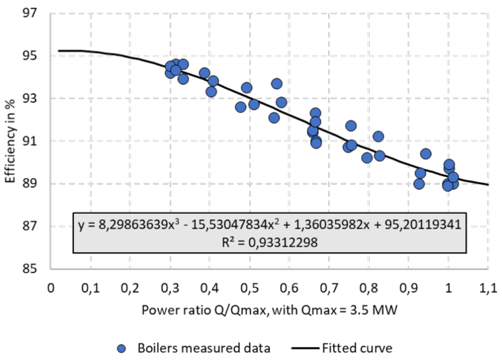


Figure 3. Measured boiler efficiency and fitted curved.

Table 1. List of operating points per dataset. Operating points number refer to points listed in Table A.2.

Dataset	Operating points n°
A	4, 8, 12, 16, 20, 24, 28
B	3, 7, 11, 15, 19, 23, 27
C	3, 4, 7, 8, 11, 12, 15, 16, 19, 20, 23, 24, 27, 28
D	2, 6,10, 14, 18, 22, 26,
E	2, 3, 6, 7, 10, 11, 14, 15, 18, 19, 22, 23, 26, 27
F	1, 5, 9, 13, 17, 21, 25
G	All points except 4, 8, 12, 16, 20, 24, 28
H	1, 2, 5, 6, 9, 10, 13, 14, 17, 18, 21, 22, 25, 26

until an "optimum" parametrization that minimize the error between model output and given operational conditions is found. The available version of the buildings' heat pump model is ready to be used with the refrigerant R401. The chiller TCHVBZ 31630 BT uses R134a as a refrigerant. Following the same approach used in Buildings.Media.Refrigerants.R410A, an implementation for R134a based on data from (Chemour 2021) is added in Modelica and Python. In regard of the parametrization, up to 28 different operating conditions were supplied by the manufacturer for this chiller. The information is summarized in the Appendix. The parametrization process is sensitive to the data used. In some case the Python script does not manage to find a proper parametrization. This is the case when all 28 operating points or data related to very different load conditions (e.g. full-100 % and low load-25 % operation) is used. A list of datasets that were used as input for the Python script that successfully outputted a "optimum" parameter set are listed in Table 1.

After the script is run for the datasets A to H, the obtained parameter sets are cross checked with all 28 operating points. The heat pump is setup in a way that the inflow temperatures at evaporator and condenser side as well as the mass flow rate trough the evaporator correspond to the measured data. The signal compressor frequency y is adjusted so that the outflow temperature at the evaporator corresponds with the measured data, thus the chilled water generated should agree with the measured data. The mass flow rate at the condenser does not correspond to the measured one, it is adjusted so that the outflow temperature matches the measured data.

The results show how for the parameter sets obtained with A, B, C, E and G the chiller cannot deliver the requested heat flow rate at the evaporator, see Figure 4. In the case of the parameter sets obtained with D, F and H, the chiller is able to deliver the requested chilled water for most of the operating points.

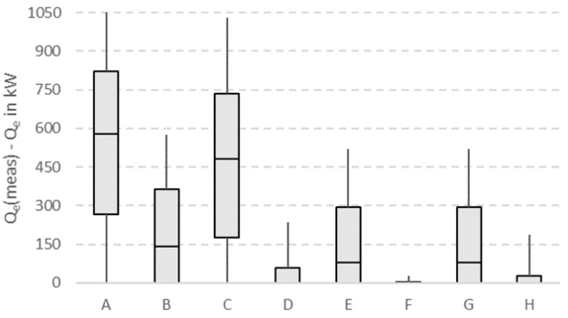


Figure 4. Deviation of heat flow rate at evaporator Q_e: Measured - model output.

From the three remaining parametrization sets, D shows very good results for most of operating points but not for full load operation. F and H shows better results for full load operation but higher deviations for most of the other operating points, see Figure 5. It is clear that if the datasets used do not include operating points on full load, the obtained parameter set or rather the parametrized model will not be able to predict correctly the real system under such conditions. However, the chiller does not operate at full load. Because it is by now not planned to operate the chiller in very different conditions than it is being operated, e.g. full load, the use of parametrization "D" suits better our needs. In regard of the electrical consumption, result obtained based on dataset D shows an average deviation of +7 % with a minimum (overprediction) and maximum (underprediction) deviation of -7 % (point n°16) and 23 % (high load operation) respectively.

3.3 Model validation

The energy system is been monitored in detail and most of the data are recorded. However, there is in some cases not enough data (due to missing measurement equipment or problems in the measuring device) to perform an energy or mass balance of some subsystems, this is the case for e.g. the boiler plant or the cold and warm water storage. In this regard, the validation work have been focused on the chiller. The same model used for the parametrization check is here used, the only difference is that measured data are used instead of single operating points. A time period of six days, from 7th until 13th of Februar 2021, is here presented.

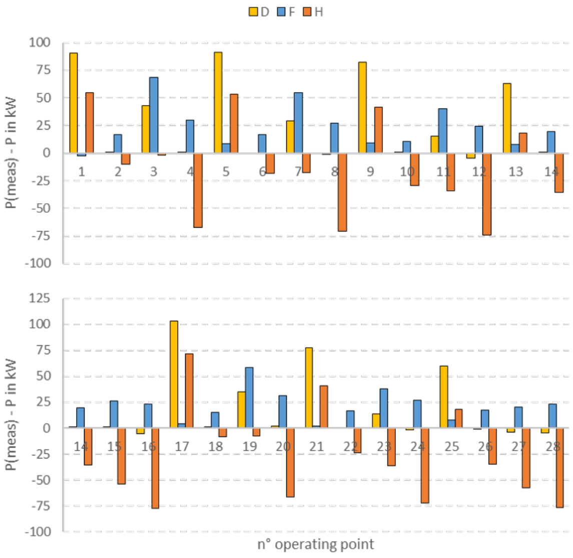


Figure 5. Deviation of electrical consumption P: Measured - model output.

The amount of chilled water as well as operating temperatures are the same between model and measured data. An overview of the temperature operating range is shown in Figure 6. The chiller is working at low load (115 kW of cooling capacity in average with a maximum and minimum of 182 kW and 71 kW respectively) and supplying chilled water at low temperature ≈ 7 °C.

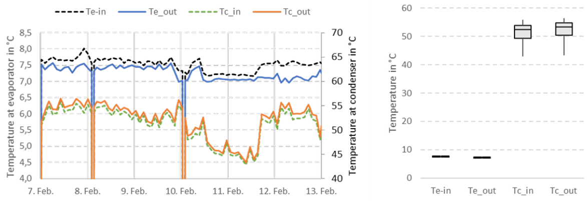


Figure 6. Operating temperatures for chiller during validation time period.

The deviations between measured data and model results can be observed in the electrical consumption and heat flow rate at the condenser and thus, EER. The differences are shown in Figures 7 to 9.

The results show how the model predicts a slightly higher but similar electrical consumption than the real system, being the average for the model and measured 62.8 kW and 60.9 kW respectively. These good agreement in the power consumption yields a similar EER, 1.88 for the model and 1.93 measured. Here is to point out that

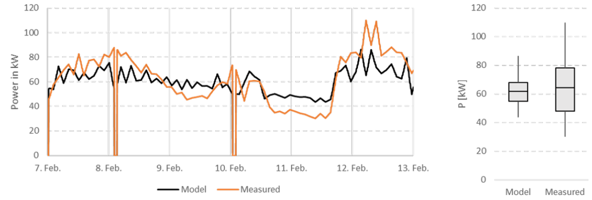


Figure 7. Electrical consumption P (measured and model output).

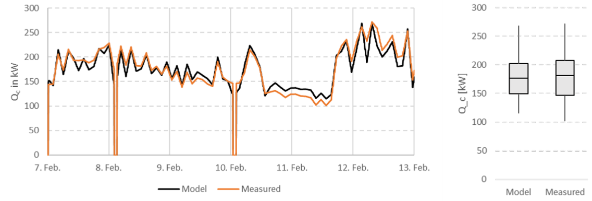


Figure 8. Heat flow rate at condenser Q_c (measured and model output).

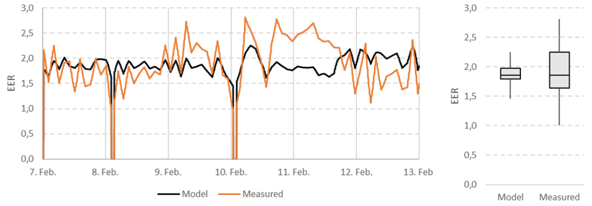


Figure 9. Energy efficiency ratio (measured and model output).

the operating conditions are similar to the operating point number 20 (see Table A.2). Operating conditions under which the model oversestimates the power consumption by less than 2 %.

3.4 Scenario evaluation

The evaluation of the scenarios is due to limited measured data done for a period of time equal to 150 days, roughly five months.

Notice that since the exact costs of water overflow, fresh water inflow, and dry coolers cannot yet be in detail evaluated (details of the system missing), thus the added value of the scenarios is limited to the main production units (boiler and chiller) as well as the amount of water pumped. The following main KPIs are considered:

- Heat supplied to the LT grid by boiler plant in MWh,
- Waste heat recovery from chiller in MWh,
- Share of heat demand at LT grid supplied by waste heat in %,
- Chiller electricity consumption in MWh,
- Fresh water into cold water storage in m^3,
- Overflowed water at warm water storage m^3.

4 Results and Discussion

Because the HT grid is merely supplied by the boiler plant, this is not of interest for the scenario evaluation and discussion and thus, not included here. In regard of the heat supply, once the waste heat of the chiller is integrated into the LT grid, the amount of heat been supplied by the boiler plant can be reduced from 1,053 MWh down to 104 MWh. The rest of the heat demand (90.1% of the overall LT grid heat demand) is covered by waste heat from the chillers, see Figure 10.

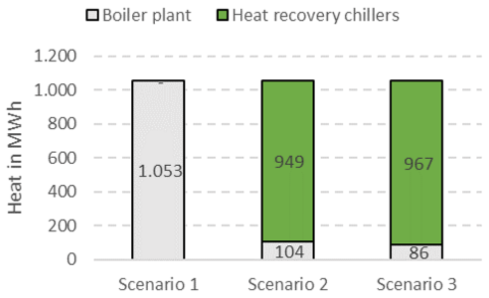


Figure 10. Share of heat supplied to cover the LT grid heat demand.

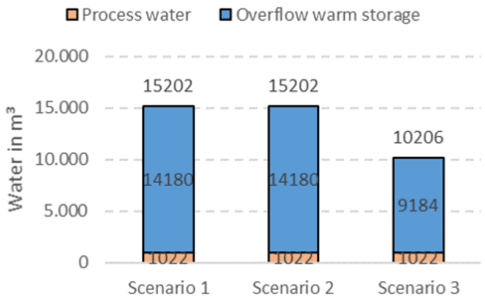


Figure 11. Total amount of fresh water needed divided into process water and overflow.

The operation conditions of the chiller in scenario 1 and 2 are almost the same. The same amount of chilled water (1,543 MWh) is produced. There is a slight difference on the temperatures at the condenser side when heat is been supplied to the low temperature grid that yield a slight variation on the electricity consumption, being in scenario 2 -1.4 % lower than scenario 1.

The amount of fresh water needed remains the same, 15,202 m^3. Most of it is pumped back into the river (14,180 m^3), and the rest (1,022 m^3) is used as process water, see Figure 11. The amount of water used to keep the temperature at the cold water storage above its minimum allowed temperature of 11 °C decreases with time due to an increase on the river temperature, see Figure 12.

The fact that not all the LT grid heat demand is covered by the waste heat of the chiller and the high amount of wa-

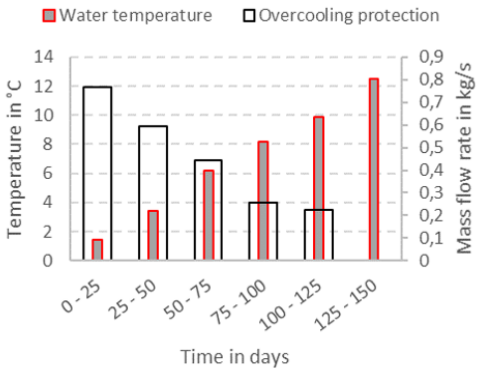


Figure 12. 25 days averaged values of river temperature and mass flow rate from warm to cold water storage (overcooling protection).

ter been pumped back into the river (14,180 m^3) motivates scenario 3.

The amount of additional chilled water generated in scenario 3 (112 MWh) is relatively low compared to the overall amount of chilled water supplied to the main cooling demand (1,543 MWh). It represents only 6.8 % of the overall chilled water in MWh. The change in the electricity consumption and amount of waste heat are low, both slightly increasing. As a result, the share of the LT grid heat demand supplied by waste heat is increased from 90.1 % in scenario 2 up to 91.8 %. The electricity consumption is 390 MWh (6.6 % higher than scenario 2).

The main added value of scenario 3 is related to the reduction of the amount of fresh water that is pumped into the cold water storage, which is specifically reduced by 32.9 % compared to scenario 1 and 2, i.e., down to 10206 m^3. Furthermore, the amount of water that have to be pumped back into the river, is also reduced by 35.2 % respect to the scenario 1 and 2 down to 9184 m^3.

5 Conclusion

Though the digitalization is often directed towards the creation of Digital Twins, the industry still lacks of groundwork, e.g., Digital Models. The amount of work need to create a Digital Model of a factory is high. Furthermore, it needs of expertise, in regard of the modeling task itself as well as of the knowledge of the system. The much-desired digitalization needs a positive balance between added value and effort. Thus, the use of the models needs to be maximized by using it e.g. to assist on regular tasks such as comparative analysis of potential improvements of the system and evaluation of measures taken (as here presented). In this regard, the improvements done in the energy supply system could be fairly compared with the former system. The results show the benefits of the chiller's waste heat integration as well as the generation of additional chilled water. In regard of the model, highlight that the heat pump model can predict accurately the power consumption of the real chiller, though the results are very sensitive to the data used on the parametrization and the choose of a proper parametrization depends on the real operation conditions and planned studies.

Acknowledgements

The research leading to these results has received funding from the Austrian Climate and Energy Fund Programme Energy Research (e!MISSION) under FFG project no. 873599.

References

AT&S (2021). *AT&S Nachhaltigkeit*. https : / / ats . net / de / unternehmen/corporate-social-responsibility/.

Brück, Dag et al. (2002). "Dymola for multi-engineering modeling and simulation". In: *2nd International Modelica Conference*.

Chemour (2021). *Freon™ 134a (R-134a)*. https://www.freon.de/products/refrigerants/r134a. Chemour.

European-Commission (2019). *Assessment of the draft National Energy and Climate Plan of Austria*. Tech. rep. European-Commission.

Fritzson, Peter and Vadim Engelson (1998). "Modelica—A Unified Object-Oriented Language for System Modeling and Simulation". In: *European Conference on Object-Oriented Programming*. Springer, pp. 67–90.

Ghobakhloo, Morteza (2020). "Industry 4.0, digitization, and opportunities for sustainability". In: *Journal of Cleaner Production* 252, p. 119869. ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2019.119869.

KG, Eberle Automatische Systeme GmbH & Co (2021). *Virtual 3D Systems*. https://www.v3s.at/de/.

Kritzinger, Werner et al. (2018). "Digital twin in manufacturing: A categorical literature review and classification". In: *IFAC-PapersOnLine* 51.11. 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018, pp. 1016–1022. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2018.08.474.

Lee, Edward A. and Sanjit A. Seshia (2017). *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. Mit Press. ISBN: 978-0-262-53381-2.

Lee, Jay, Behrad Bagheri, and Hung-An Kao (2015). "A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems". In: *Manufacturing Letters* 3, pp. 18–23. ISSN: 2213-8463. DOI: https://doi.org/10.1016/j.mfglet.2014.12.001.

Luderer, Gunnar et al. (2018). "Residual fossil CO 2 emissions in 1.5–2 C pathways". In: *Nature Climate Change* 8.7, pp. 626–633.

Philibert, Cédric (2017). "Renewable Energy for Industry: From Green Energy to Green Materials and Fuels". In: *Paris: International Energy Agency*.

Schweiger, Gerald, Georg Engel, et al. (2018). "Co-simulation–an empirical survey: applications, recent developments and future challenges". In: *MATHMOD 2018 Extended Abstract Volume*, pp. 125–126.

Schweiger, Gerald, Henrik Nilsson, et al. (2020). “Modeling and simulation of large-scale systems: A systematic comparison of modeling paradigms”. In: *Applied Mathematics and Computation* 365, p. 124713.

Tao, Fei et al. (2019). “Digital Twins and Cyber–Physical Systems toward Smart Manufacturing and Industry 4.0: Correlation and Comparison”. In: *Engineering* 5.4, pp. 653–661. ISSN: 2095-8099. DOI: https://doi.org/10.1016/j.eng.2019.01.014.

Wetter, Michael, Christoph van Treeck, et al. (2019). “IBPSA Project 1: BIM/GIS and Modelica framework for building and community energy system design and operation–ongoing developments, lessons learned and challenges”. In: *IOP Conference Series: Earth and Environmental Science*. Vol. 323. IOP Publishing, p. 012114.

Wetter, Michael, Wangda Zuo, et al. (2014). “Modelica Buildings library”. In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506.

A Appendix

Table A.2. Detailed list of operating points for TCHVBZ 31630 BT

n°	T_{c-in} [°C]	T_{e-in} [°C]	Q_e [kW]	Q_c [kW]	P [kW]
1	40	7	1,150	1,568	418
2	42	7	959	1,295	336
3	43	7	684	953	269
4	46	7	321	434	113
5	35	7	1,240	1,633	392
6	36	7	1,023	1,331	308
7	38	7	739	981	241
8	41	7	343	446	103
9	29	7	1,346	1,705	359
10	31	7	1,109	1,393	284
11	33	7	805	1,021	217
12	36	7	371	467	95
13	19	7	1,467	1,766	299
14	21	7	1,218	1,460	242
15	23	7	904	1,079	175
16	26	7	408	489	81
17	39	10	1,294	1,736	442
18	41	10	1,066	1,409	343
19	43	10	767	1,038	270
20	46	10	357	472	115
21	29	10	1,452	1,817	366
22	31	10	1,202	1,492	290
23	33	10	883	1,099	217
24	36	10	403	500	97
25	19	10	1,593	1,899	306
26	20	10	1,343	1,591	248
27	22	10	1,002	1,178	176
28	25	10	450	533	83

Handling Multimode Models and Mode Changes in Modelica

Albert Benveniste[1] Benoît Caillaud[1] Mathias Malandain[1]

[1]Inria Centre de Rennes Bretagne Atlantique, University of Rennes 1, France,
{albert.benveniste,benoit.caillaud,mathias.malandain}@inria.fr

Abstract

Since its version 3.3, the Modelica language offers the possibility to model multimode systems having different DAE-based dynamics in each mode, thanks to the introduction of state machines. When the differentiation index and structure varies with mode changes, compilers generate erroneous simulation code, often resulting in runtime exceptions. We propose in this paper a multimode structural analysis for both multiple modes and mode change events and we show how correct code for restarts can be generated. Our approach is illustrated on two simple but representative mechanical systems.

Keywords: multimode DAE, structural analysis

1 Introduction

Since version 3.3, the Modelica language offers the possibility of specifying *multimode dynamics,* by describing state machines with different DAE dynamics in each different state (Elmqvist, Gaucher, et al. 2012). This feature enables describing large complex cyber-physical systems with different behaviors in different modes.

While being undoubtedly valuable, multimode modeling has been the source of serious difficulties for non-expert users of the current generation of Modelica tools. Indeed, while many large-scale Modelica models are properly handled, some physically meaningful models do not result in correct simulations with most Modelica tools. As such problematic models are actually easy to construct, the likelihood of such bad cases occurring in large models is significant.

It is unfortunately unclear which multimode Modelica models will be properly handled, and which ones will fail. As a consequence, quite often, end users have to ask Modelica experts, or even tool developers themselves, to tweak their models in order to make them work as expected. While it is accepted that physical modeling itself requires expertise, requiring expertise in how to get around tool idiosyncrasies is not desirable. This situation hinders a wider spreading of Modelica tools among a larger class of users, such as Simulink-trained engineers.

As our review of two examples will reveal, this problem is mainly due to an inadequate structural analysis, performed during compilation. As far as we know, no industrial-strength Modelica tool implements a mode-dependent structural analysis—a few academic prototypes address this difficulty in part, see Section 3. Worse, it is not even understood what kind of structural analysis should be associated with mode change events.

Some years ago, we started a project aiming at addressing all the above issues. In this paper we explain our approach, by illustrating it on two simple yet physically meaningful examples that current Modelica tools fail to properly simulate. The use of nonstandard analysis allows us to perform the analysis of both modes and mode changes in a unified framework, including the handling of transient modes and that of impulsive mode changes. Standardization techniques are then used in order to generate effective code for restarts at mode changes. As an efficient implementation of such methods in Modelica compilers would greatly expand the class of multimode models amenable to reliable numerical simulation, we hint at possible mechanizations towards the end of the paper; this aspect is developed in both the companion paper (Benveniste, Caillaud, and Malandain 2021). and the previously published article (Caillaud, Malandain, and Thibault 2020).

2 Two problematic examples

We review two small examples of multimode DAE systems and analyse how they are handled by two state-of-the-art Modelica tools, OpenModelica and Dymola.

2.1 An ideal clutch

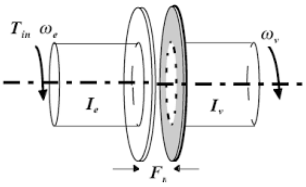


Figure 1. An ideal clutch with two shafts.

The clutch depicted in Figure 1 is an idealized clutch interconnecting two rotating shafts. It is assumed that this system is closed, meaning that the two shafts are not connected to anything else, whence the corresponding model:

$$\left\{\begin{array}{rllr} & & \omega_1' = f_1(\omega_1, \tau_1) & (e_1) \\ & & \omega_2' = f_2(\omega_2, \tau_2) & (e_2) \\ \texttt{if}\ \gamma & \texttt{do} & \omega_1 - \omega_2 = 0 & (e_3) \\ & \texttt{and} & \tau_1 + \tau_2 = 0 & (e_4) \\ \texttt{if not}\ \gamma & \texttt{do} & \tau_1 = 0 & (e_5) \\ & \texttt{and} & \tau_2 = 0 & (e_6) \end{array}\right. \quad (1)$$

In model (1), the dynamics of each shaft i is described by ODE $\omega_i' = f_i(\omega_i, \tau_i)$ for some, yet unspecified, function f_i, where ω_i is the angular velocity and τ_i is the torque applied to shaft i. Depending on the value of the input Boolean variable γ, the clutch is either engaged ($\gamma = \mathrm{T}$, the constant "true") or released ($\gamma = \mathrm{F}$, the constant "false"). When the clutch is released, the two shafts rotate freely: no torque is applied to them ($\tau_i = 0$). When the clutch is engaged, it ensures a perfect join between the two shafts, forcing them to have the same angular velocity ($\omega_1 - \omega_2 = 0$) and opposite torques ($\tau_1 + \tau_2 = 0$). When $\gamma = \mathrm{T}$, equations (e_3, e_4) are active and equations (e_5, e_6) are disabled, and vice-versa when $\gamma = \mathrm{F}$. If the clutch is initially released, then, at the instant of contact, the relative speed of the two rotating shafts jumps to zero; as a consequence, an impulse is expected on the torques.

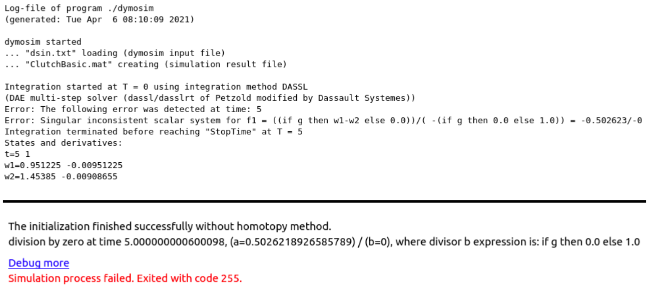
The model yields an ODE system when the clutch is released, and a DAE system of index 1 when the clutch is engaged (see Section 5.1).

The clutch in Modelica: Figure 2 details the Modelica model of the Ideal Clutch system. It is a faithful translation in the Modelica language of the two-mode DAE (1), except that the two differential equations have been linearized. Also, the trajectory of the input guard γ (here called `g`) has been fully specified: it takes the value T between t_1 and t_2 and F otherwise.

```
model ClutchBasic
parameter Real w01=1;
parameter Real w02=1.5;
parameter Real j1=1;
parameter Real j2=2;
parameter Real k1=0.01;
parameter Real k2=0.0125;
parameter Real t1=5;
parameter Real t2=7;
Real t(start=0, fixed=true);
Boolean g(start=false);
Real w1(start = w01, fixed=true);
Real w2(start = w02, fixed=true);
Real f1;  Real f2;
equation
  der(t) = 1;
  g = (t >= t1) and (t <= t2);
  j1*der(w1) = -k1*w1 + f1;
  j2*der(w2) = -k2*w2 + f2;
  0 = if g then w1-w2 else f1;
  f1 + f2 = 0;
end ClutchBasic;
```

Figure 2. Modelica code for the idealized clutch.

This model is deemed structurally nonsingular by the two Modelica tools we had the opportunity to test: OpenModelica 1.17.0 (Fritzson et al. 2020) and Dymola 2021 (Dassault Systèmes AB 2020). However, none of these tools generates correct simulation code from this model. Indeed, simulations fail precisely at the instant when the clutch switches from the uncoupled mode (`g=false`) to the coupled one (`g=true`). This is evidenced by a division by zero exception, as shown in Figure 3.



```
Log-file of program ./dymosim
(generated: Tue Apr  6 08:10:09 2021)

dymosim started
... "dsin.txt" loading (dymosim input file)
... "ClutchBasic.mat" creating (simulation result file)

Integration started at T = 0 using integration method DASSL
(DAE multi-step solver (dassl/dasslrt of Petzold modified by Dassault Systemes))
Error: The following error was detected at time: 5
Error: Singular inconsistent scalar system for f1 = ((if g then w1-w2 else 0.0))/( -(if g then 0.0 else 1.0)) = -0.502623/-0
Integration terminated before reaching "StopTime" at T = 5
States and derivatives:
t=5 1
w1=0.951225 -0.00951225
w2=1.45385 -0.00908655
```

```
The initialization finished successfully without homotopy method.
division by zero at time 5.000000000600098, (a=0.5026218926585789) / (b=0), where divisor b expression is: if g then 0.0 else 1.0
Debug more
Simulation process failed. Exited with code 255.
```

Figure 3. Division by zero exceptions with Dymola 2021 (top) and OpenModelica 1.17.0 (bottom) occuring when simulating the Ideal Clutch Modelica model.

The cause of this exception is that none of these tools performs a multimode structural analysis. Instead, the structure of the model is assumed invariant, and a Dummy Derivatives method (Mattsson and Soderlind 1993) is implemented, which is correct on single-mode DAE systems, whereas it may fail on multimode systems unless the model structure is independent of the mode. The structural analysis methods in these tools do not detect that the differentiation index jumps from 0 to 1 when the shafts are coupled, and that the structure is not invariant. The division by zero results from the pivoting of a linear system of equations that becomes singular when `g` becomes equal to `true`.

2.2 A Cup-and-Ball game

Figure 4. The Cup-and-Ball game.

We sketch here a multimode extension of the popular example of the pendulum in Cartesian coordinates (Pantelides 1988), namely the Cup-and-Ball game illustrated by Figure 4. A ball, modeled by a point mass, is attached to one end of a rope, while the other end of the rope is fixed, to the origin of the plane in the model. The ball is subject to the unilateral constraint set by the rope, but moves freely while the distance between the ball and the origin is less than its length. The system is assumed closed. The model for a 2D-version of this example is:

$$\begin{cases} 0 = x'' + \lambda x & (e_1) \\ 0 = y'' + \lambda y + g & (e_2) \\ 0 \leq L^2 - (x^2 + y^2) & (\kappa_1) \\ 0 \leq \lambda & (\kappa_2) \\ 0 = \left[L^2 - (x^2 + y^2)\right] \times \lambda & (\kappa_3) \end{cases} \qquad (2)$$

where the dependent variables are the position (x,y) of the ball in Cartesian coordinates and the rope tension λ.

The subsystem $(\kappa_1, \kappa_2, \kappa_3)$ expresses that the tension is nonnegative, the distance of the ball from the origin is less than or equal to L, and one cannot have a nonzero tension and a distance less than L at the same time. Constraints κ_1 and κ_2 are unilateral, which is not supported by Modelica and related languages. Therefore, using the technique presented in (Mattsson, Otter, and Elmqvist 1999), we redefine the graph of this complementarity condition as a parametric curve, represented by the following three equations:

$$\begin{array}{rcl} s & = & \texttt{if}\,\gamma\,\texttt{then}-\lambda\;\texttt{else}\,L^2-(x^2+y^2) \\ 0 & = & \texttt{if}\,\gamma\,\texttt{then}\,L^2-(x^2+y^2)\;\texttt{else}\,\lambda \\ \gamma & = & [s \le 0] \end{array} \qquad (3)$$

Similarly to the clutch model, impulsive behavior is expected on the torques. However, an other possible difficulty is present: subsystem $(\kappa_1, \kappa_2, \kappa_3)$ of (2) leaves the impact law at mode change insufficiently specified; it could be fully elastic, fully inelastic, or in between. *Can both of these aspects be detected at compile time, using some kind of structural analysis?*

The Cup-and-Ball in Modelica: Figure 5 details the Modelica model of the Cup-and-Ball game. It is a faithful translation of the two-mode DAE (2) using rewriting (3). The point mass, modeling the ball, initially stands at the origin of the plane with zero velocity; the Boolean guard γ, named `gamma` in the model, is thus set to `false`.

```
model CupAndBall
constant Real g=9.81;
constant Real L=1.0;
Real x(start=0,fixed=true);
Real y(start=0,fixed=true);
Real u(start=0,fixed=true);
Real v(start=0,fixed=true);
Real lambda;
Real s;
Boolean gamma(start=false,fixed=true);
equation
   der(x) = u;
   der(y) = v;
   der(u) + lambda*x = 0;
   der(v) + lambda*y + g = 0;
   gamma = (s <= 0);
   0 = if gamma then L^2 - (x^2 + y^2)
               else lambda;
   s = if gamma then - lambda
               else L^2 - (x^2 + y^2);
end CupAndBall;
```

Figure 5. Modelica code for the Cup-and-Ball.

As is the case for the clutch model presented above, this model is deemed structurally nonsingular by both OpenModelica 1.17.0 and Dymola 2021, but the simulation fails at the instant of mode change. Figure 6 depicts the resulting trajectory of variables `y` and `gamma`; it ends when `gamma` switches from `false` to `true`, as the tool is unable to correctly reinitialize the model after the mode change. Replacing condition `s <= 0` with `last(s) <= 0` in order to break the fixpoint equation defining variable `gamma` (see the introduction of Section 6) leads to the same simulation results, but with a division by zero error similar to that shown in Figure 3 occurring at the moment of mode change.

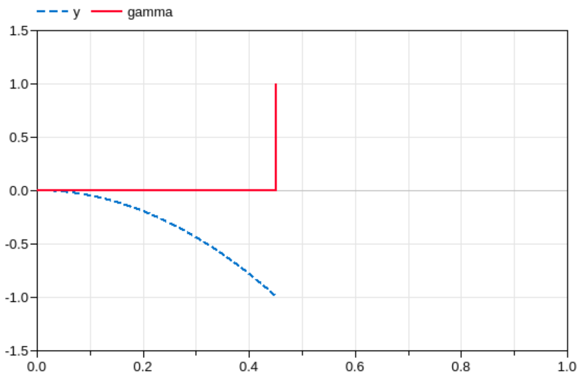


Figure 6. Trajectory of the Cup-and-Ball Modelica model: it stops around $t = 0.452s$, when the rope becomes straight.

.

It appears from both exemples that some fundamental study is needed to correctly simulate multimode models, and that the problem is twofold: the varying structure of the model has to be taken into account, and mode changes have to be handled in a specific fashion.

Smoothing '`if then else`' equations could help solve both issues by essentially turning multimode models into single-mode models, but this requires a delicate and definitely non-modular tuning, as it depends on the different time scales arising in the system. We believe that, as the tools reputedly support multimode DAE models, they should handle them correctly.

3 Related work

For the general literature on DAE and Modelica, we refer the reader to `https://www.modelica.org/publications` and (Benveniste, Caillaud, Elmqvist, et al. 2019; Benveniste, Caillaud, and Malandain 2020).

Elmqvist, Mattsson, and Otter (2014) and Mattsson, Otter, and Elmqvist (2015) propose a high-level description of multimode models as an extension to the synchronous Modelica 3.3 state machines, by using continuous-time state machines having continuous-time models as "states". State machines are transformed so that the resulting equations can be processed by standard symbolic algorithms supported by Modelica tools. Describing variable-structure systems with *causal* state machines is discussed in (Pepper et al. 2011). Dynamically changing the structural analysis at runtime is also proposed in (Höger 2014; Höger 2017), with Höger (2014) proposing a dynamic execution of the Σ-method (Pryce 2001), and by Nilsson and Giorgidze (2010) in the context of their Functional

Hybrid Modelling paradigm. Such approaches typically rely on the explicit declaration of reinitializations at mode changes.

As such, the computation of correct restarts at mode changes, while being a central issue in multi-mode DAE systems, is not being tackled in the already mentioned references. Some authors still address this issue.

Benveniste, Caillaud, Elmqvist, et al. (2017) tackled this issue, as well as the problem of varying structure and index, from a fundamental point of view, by relying on nonstandard analysis to capture continuous-time dynamics and mode change events in a unified framework. A first structural analysis algorithm was presented in this paper, by significantly modifying the original Pantelides algorithm (Pantelides 1988). This first attempt suffers from some deficiencies: the proposed structural analysis does not boil down to the Pantelides algorithm in the case of single-mode systems; it involves nondeterministic decisions, an unwanted feature for the mathematical foundation of compilers; and its mathematical study is incomplete.

(Trenn 2009b; Trenn 2009a) are important works as they point out the difficulty in defining piecewise smooth distributions. Liberzon and Trenn were able to define complete solutions for a class of switched DAE systems in which each mode is in *quasi-linear form* (Liberzon and Trenn 2012): notably, switching conditions are time-based only.

In (Benveniste, Caillaud, Elmqvist, et al. 2019), an interesting subclass of multimode DAE systems was identified, which possibly exhibit impulsive variables at mode changes. They extend the "quasi-linear systems" proposed by Trenn et al.; in particular, switching conditions are no longer restricted to be time-based, but can be state-based. Nevertheless, the analysis and discretization schemes proposed in (Benveniste, Caillaud, Elmqvist, et al. 2019) are mathematically sound. Building on this work, Elmqvist and Otter have developed the ModiaSim[1] Julia packages for semi-linear multimode DAE systems. It turns out that the general approach of the present paper coincides with the schemes proposed in (Benveniste, Caillaud, Elmqvist, et al. 2019) when applied to the considered subclass. Our present contribution thus extends and significantly improves that work. An in-depth comparison can be found in (Benveniste, Caillaud, and Malandain 2020).

4 Our contributions

Structural analysis of mode changes and code generation for restarts: *We develop a structural analysis that is valid at any time, that is, for both continuous dynamics and mode changes.* Impulsive behaviors may occur at mode changes for certain variables. Whereas the few current tools able to handle such situations discover them at runtime, our structural analysis covers these situations and handles them at compile time.

Rejecting or accepting programs on a clear basis, at compile time: Our structural analysis is precise enough to properly identify models that are structurally over- or under-specified at mode change events. In turn, mode-dependent index/state/dynamics are not reasons for rejection: our approach handles such cases.

We now move to developing our approach by discussing the two examples from Section 2.

5 The ideal clutch

Its model was given in (1). We first analyze separately the model for each mode of the clutch. Then, we focus on mode changes and propose a comprehensive analysis.

5.1 Separate Analysis of Each Mode

In the released mode, i.e., when $\gamma = \text{F}$ in System (1), the two shafts are independent and one obtains the following two independent ODEs for ω_1 and ω_2:

$$\begin{array}{llll} \omega_1' = f_1(\omega_1, \tau_1) & (e_1) & \tau_1 = 0 & (e_5) \\ \omega_2' = f_2(\omega_2, \tau_2) & (e_2) & \tau_2 = 0 & (e_6) \end{array} \qquad (4)$$

In the engaged mode, however ($\gamma = \text{T}$), the two velocities and torques are algebraically related:

$$\begin{array}{llrl} \omega_1' = f_1(\omega_1, \tau_1) & (e_1) & \omega_1 - \omega_2 = 0 & (e_3) \\ \omega_2' = f_2(\omega_2, \tau_2) & (e_2) & \tau_1 + \tau_2 = 0 & (e_4) \end{array} \qquad (5)$$

System (5) is a DAE. Its structural analysis tells that equation (e_3) must be differentiated and added to the model (it is highlighted in red):

$$\begin{array}{llrl} & & \omega_1 - \omega_2 = 0 & (e_3) \\ \omega_1' = f_1(\omega_1, \tau_1) & (e_1) & \omega_1' - \omega_2' = 0 & (e_3') \\ \omega_2' = f_2(\omega_2, \tau_2) & (e_2) & \tau_1 + \tau_2 = 0 & (e_4) \end{array} \qquad (6)$$

Although this change of differentiation index is the root cause of the runtime exceptions shown in Figure 3, solving this issue would not be enough for the correct simulation of the model, because of the need of handling mode changes.

As a matter of fact, while the cold initialization of the engaged mode yields 6 dependent variables for only 5 equations, thus leaving one degree of freedom (the common velocity of the two shafts), the mode change $\gamma : \text{F} \rightarrow \text{T}$, when the clutch gets engaged, is physically determinate, which makes the point that mode changes cannot be handled as "cold restarts".

Inferring by hand the reset values for rotation velocities when the clutch gets engaged is definitely non-trivial. Furthermore, these values depend on the whole system model, so that the task of determining them becomes complex if external components are added.

***It is therefore highly desirable, for this example, to let the compiler infer these reset values from model* (1).**

[1] https://modiasim.github.io/docs/index.html

5.2 Mapping model to nonstandard analysis

If DAE dynamics is approximated in discrete time, then the whole model becomes discrete-time. To avoid the problem of approximation error, our idea is to use an "infinitesimal" time step in the discrete time approximation. This will yield an approximation up to an infinitesimal accuracy.

This can be made rigorous by relying on *nonstandard analysis* (Robinson 1996; Lindstrøm 1988; Benveniste, Caillaud, and Malandain 2020), which extends the set $\mathbb{R}$ of real numbers to a superset ${}^\star\mathbb{R}$ of *hyperreals* that includes infinite sets of infinitely large numbers and infinitely small numbers.

For the understanding of this paper, it is enough to know the following about nonstandard analysis. There exist *infinitesimals*, defined as hyperreals that are smaller in absolute value than any real number. The arithmetic operations $+$, $\times$, etc., and usual relations, are lifted to ${}^\star\mathbb{R}$. For every finite hyperreal $x \in {}^\star\mathbb{R}$, there is a unique standard real number $\mathtt{st}(x) \in \mathbb{R}$ such that $\mathtt{st}(x) - x$ is infinitesimal, and $\mathtt{st}(x)$ is called the *standard part* (or *standardization*) of x. Standardizing functions or systems of equations, however, requires some care. One important issue is derivatives. For $t \mapsto x(t)$ an $\mathbb{R}$-valued (standard) signal ($t \in \mathbb{R}$),

x is differentiable at instant $t{\in}\mathbb{R}$ if and only if there exists $a{\in}\mathbb{R}$ such that, for any infinitesimal $\partial \in {}^\star\mathbb{R}$, $\frac{x(t+\partial)-x(t)}{\partial} - a$ is infinitesimal; then, $a = x'(t)$. (7)

We can then consider the time index set $\mathbb{T} \subseteq {}^\star\mathbb{R}$:

$$\mathbb{T} = 0, \partial, 2\partial, 3\partial, \cdots = \{n\partial \mid n \in {}^\star\mathbb{N}\} \tag{8}$$

where ${}^\star\mathbb{N}$ denotes the set of *hyperintegers*, consisting of all integers augmented with additional infinite numbers called *nonstandard*, and ∂ is an arbitrary, but fixed, infinitesimal.[2] The following features of $\mathbb{T}$ are important: (1) any finite real time $t{\in}\mathbb{R}$ is infinitesimally close to some element of $\mathbb{T}$ (hence, $\mathbb{T}$ covers $\mathbb{R}$ and can be used to index continuous-time dynamics); and (2) $\mathbb{T}$ is "discrete": every instant $n\partial$ has a predecessor $(n{-}1)\partial$ (except for $n = 0$) and a successor $(n{+}1)\partial$.

Let x be a nonstandard signal indexed by $\mathbb{T}$. The *forward-* and *backward-shifted* signals $x^\bullet$ and ${}^\bullet x$ are defined by:

$$x^\bullet(n\partial) =_{\text{def}} x((n{+}1)\partial) \;\text{ and }\; {}^\bullet x((n{+}1)\partial) =_{\text{def}} x(n\partial)\,,$$

implying that an initial value for ${}^\bullet x(0)$ must be provided. For $f(X)$ a function of the tuple X of signals, we set $(f(X))^\bullet =_{\text{def}} f(X^\bullet)$ where the forward shift $X \mapsto X^\bullet$ applies pointwise to all the components of the tuple. For example, $f^\bullet(x,y)(t) = f(x^\bullet, y^\bullet) = f(x(t{+}\partial), y(t{+}\partial))$.

Using (7), we represent, up to an infinitesimal, the derivative x' of a signal by its first-order explicit Euler approximation $\frac{1}{\partial}(x^\bullet - x)$. Solutions of multi-mode DAE systems may be non-differentiable or even non-continuous at events of mode change. To give a meaning to x' at any instant, *we **define** it everywhere as*

$$x' =_{\text{def}} \frac{1}{\partial}(x^\bullet - x)\,. \tag{9}$$

The nonstandard expansion of two-mode system (4,6) is:

$$\left\{\begin{array}{rlll}
 & & \frac{\omega_1^\bullet - \omega_1}{\partial} = f_1(\omega_1, \tau_1) & (e_1^\partial) \\
 & & \frac{\omega_2^\bullet - \omega_2}{\partial} = f_2(\omega_2, \tau_2) & (e_2^\partial) \\
\texttt{if}\ \gamma & \texttt{do} & \omega_1 - \omega_2 = 0 & (e_3) \\
 & \texttt{and} & \omega_1^\bullet - \omega_2^\bullet = 0 & (e_3^\bullet) \\
 & \texttt{and} & \tau_1 + \tau_2 = 0 & (e_4) \\
\texttt{if not}\ \gamma & \texttt{do} & \tau_1 = 0 & (e_5) \\
 & \texttt{and} & \tau_2 = 0 & (e_6)
\end{array}\right. \tag{10}$$

Note that the latent differentiated equation (e_3') of model (6) has been replaced by the forward shifted equation $(e_3^\bullet)$ (both are equivalent from a structural point of view). The state variables are ω_1, ω_2 whereas the leading variables are now τ_1, τ_2, $\omega_1^\bullet$, $\omega_2^\bullet$, in both modes $\gamma = \mathrm{F}$ and $\gamma = \mathrm{T}$. This yields a sort of explicit Euler scheme for model (1), which is exact up to infinitesimals within each mode. The structural analysis is correct in each mode.

5.3 Structural analysis of mode change γ:F→T

We focus on mode change $\gamma : \mathrm{F} \to \mathrm{T}$, when the clutch gets engaged. At the considered instant, we have ${}^\bullet\gamma = \mathrm{F}$ and $\gamma = \mathrm{T}$. We unfold System (10) at the two successive (previous and current) instants by taking the actual values for the guard at those instants into account:

$$\begin{array}{rl}
\begin{array}{r}\text{previous}\\ \text{instant}\\ \gamma = \mathrm{F}\end{array} & \left\{\begin{array}{ll}
\frac{\omega_1 - {}^\bullet\omega_1}{\partial} = f_1({}^\bullet\omega_1, {}^\bullet\tau_1) & ({}^\bullet e_1^\partial) \\
\frac{\omega_2 - {}^\bullet\omega_2}{\partial} = f_2({}^\bullet\omega_2, {}^\bullet\tau_2) & ({}^\bullet e_2^\partial) \\
{}^\bullet\tau_1 = 0 & \\
{}^\bullet\tau_2 = 0 &
\end{array}\right. \\
\begin{array}{r}\text{current}\\ \text{instant}\\ \gamma = \mathrm{T}\end{array} & \left\{\begin{array}{ll}
\frac{\omega_1^\bullet - \omega_1}{\partial} = f_1(\omega_1, \tau_1) & \\
\frac{\omega_2^\bullet - \omega_2}{\partial} = f_2(\omega_2, \tau_2) & \\
\omega_1 - \omega_2 = 0 & (e_3) \\
\omega_1^\bullet - \omega_2^\bullet = 0 & \\
\tau_1 + \tau_2 = 0 &
\end{array}\right.
\end{array} \tag{11}$$

We regard System (11) as an algebraic system of equations with dependent variables ${}^\bullet\tau_i, \omega_i; \tau_i, \omega_i^\bullet$ for $i = 1, 2$, i.e., the leading variables of System (10) at the previous and current instants. System (11) is structurally singular, as it includes the following subsystem[3] which has five equations

[2] It is proved in (Benveniste, Caillaud, and Malandain 2020) that the simulation code that is finally generated does not depend on the choice of this infinitesimal time step.

[3] Over- and underdetermined subsystems are structurally found by computing the Dulmage-Mendelsohn decomposition of the system (Dulmage and Mendelsohn 1958).

and only four dependent variables $\omega_1, \omega_2, {}^\bullet\tau_1, {}^\bullet\tau_2$:

$$\left\{\begin{array}{ll} \frac{\omega_1 - {}^\bullet\omega_1}{\partial} = f_1({}^\bullet\omega_1, {}^\bullet\tau_1) & ({}^\bullet e_1^\partial) \\ \frac{\omega_2 - {}^\bullet\omega_2}{\partial} = f_2({}^\bullet\omega_2, {}^\bullet\tau_2) & ({}^\bullet e_2^\partial) \\ {}^\bullet\tau_1 = 0 & \\ {}^\bullet\tau_2 = 0 & \\ \omega_1 - \omega_2 = 0 & (e_3) \end{array}\right. \tag{12}$$

We resolve this conflict by applying the following principle:

Principle 1 (causality) *What was done at the previous instant cannot be undone at the current instant.*

Applying (1) leads to removing, from subsystem (12), the conflicting equation (e_3). This yields the following nonstandard code for the restart at mode change $\gamma : \mathrm{F} \to \mathrm{T}$:

$$\left\{\begin{array}{l} \omega_1, \omega_2, {}^\bullet\tau_1, {}^\bullet\tau_2 \text{ set by previous instant} \\ \omega_1^\bullet = \omega_1 + \partial \times f_1(\omega_1, \tau_1) \\ \omega_2^\bullet = \omega_2 + \partial \times f_2(\omega_2, \tau_2) \\ \omega_1^\bullet - \omega_2^\bullet = 0 \\ \tau_1 + \tau_2 = 0 \end{array}\right. \tag{13}$$

The consistency equation $(e_3) : \omega_1 - \omega_2 = 0$ has been removed from System (13), thus modifying the original model. However, this removal occurs only at mode change events $\gamma : \mathrm{F} \to \mathrm{T}$. What we have done amounts to *delaying by one nonstandard instant the satisfaction of some of the constraints in force in the new mode $\gamma = \mathrm{T}$.* Since our time step ∂ is infinitesimal, this takes zero standard time.

5.4 Generating effective code for restart at mode change $\gamma : \mathrm{F} \to \mathrm{T}$

We wish to use System (13) by identifying current values for the states ω_i with the *left-limits* ω_i^- i.e., the values of the velocities just before the mode change. From these values, we would then compute the restart values for the velocities $\omega_i^+ =_{\text{def}} \omega_i^\bullet$, together with the torques τ_i.

Unfortunately, hyperreals are unknown to computers, hence, System (13) cannot be used as such, but needs to be *standardized,* by “washing out” ∂. Since the time step ∂ is infinitesimal, it is tempting to get rid of of it in (13) by simply setting $\partial = 0$. Unfortunately, doing this leaves us with a structurally singular system, since the two torques are then involved in only one equation.

This problem of structural singularity is in fact due to the existence of impulsive variables. To discover them in a systematic way, we perform an *impulse analysis*.

Impulse analysis: Before engaging the clutch, we must generically assume $\omega_1 - \omega_2 \neq 0$. Since $\omega_1^\bullet - \omega_2^\bullet = 0$ holds, $\frac{(\omega_1^\bullet - \omega_2^\bullet) - (\omega_1 - \omega_2)}{\partial} = f_1(\omega_1, \tau_1) - f_2(\omega_2, \tau_2)$ cannot be finite because, if it was, then the function $\omega_1 - \omega_2$ would be continuous, contradicting the assumption that $\omega_1 - \omega_2 \neq 0$. Hence, the hyperreal $f_1(\omega_1, \tau_1) - f_2(\omega_2, \tau_2)$ is necessarily infinite. However, we assumed continuous functions f_i and finite state (ω_1, ω_2). Thus, one of the torques τ_i must be infinite at mode change, and because of equation $(e_4) : \tau_1 + \tau_2 = 0$, both torques are in fact infinite, i.e., are *impulsive*.

Eliminating impulsive variables: We now assume that the f_i's are linear in the torques, i.e., each f_i has the form

$$f_i(\omega_i, \tau_i) \quad = \quad a_i(\omega_i) + b_i(\omega_i)\tau_i \,, \tag{14}$$

where b_1 and b_2 are the inverse moments of inertia of the rotating masses and a_1 and a_2 are damping factors divided by the corresponding moments of inertia. This yields the following system of equations, to be solved for $\omega_1^\bullet, \omega_2^\bullet, \tau_1, \tau_2$ at the instant when γ switches from F to T:

$$\left\{\begin{array}{ll} \omega_1^\bullet = \omega_1 + \partial\,(a_1(\omega_1) + b_1(\omega_1)\tau_1) & (e_1^\partial) \\ \omega_2^\bullet = \omega_2 + \partial\,(a_2(\omega_2) + b_2(\omega_2)\tau_2) & (e_2^\partial) \\ \omega_1^\bullet - \omega_2^\bullet = 0 & (e_3^\bullet) \\ \tau_1 + \tau_2 = 0 & (e_4) \end{array}\right. \tag{15}$$

We now eliminate the impulsive variables from System (15), namely, the two torques. Using (e_4) yields $-\tau_2 = \tau_1 =_{\text{def}} \tau$. Premultiplying the system of equations

$$\left\{\begin{array}{ll} \omega_1^\bullet = \omega_1 + \partial\,(a_1(\omega_1) + b_1(\omega_1)\tau) & (e_1^\partial) \\ \omega_2^\bullet = \omega_2 + \partial\,(a_2(\omega_2) - b_2(\omega_2)\tau) & (e_2^\partial) \end{array}\right.$$

by the row matrix $\left[\begin{array}{cc} b_2(\omega_2) & b_1(\omega_1) \end{array}\right]$ yields

$$\begin{array}{l} b_2(\omega_2)\,\omega_1^\bullet + b_1(\omega_1)\,\omega_2^\bullet = \\ b_2(\omega_2)\,(\omega_1 + \partial\, a_1(\omega_1)) + b_1(\omega_1)\,(\omega_2 + \partial\, a_2(\omega_2))\,. \end{array}$$

Using in addition $(e_3^\bullet)$ and setting $\omega^\bullet =_{\text{def}} \omega_1^\bullet = \omega_2^\bullet$ yields

$$\begin{aligned} \omega^\bullet \quad = \quad & \frac{b_2(\omega_2)\omega_1 + b_1(\omega_1)\omega_2}{b_1(\omega_1) + b_2(\omega_2)} \\ & + \partial\,\frac{a_1(\omega_1)b_2(\omega_2) + a_2(\omega_2)b_1(\omega_1)}{b_1(\omega_1) + b_2(\omega_2)} \end{aligned} \tag{16}$$

It is now legitimate to set $\partial = 0$ in its right-hand side. This yields, by identifying $\mathsf{st}(\omega_i) = \omega_i^-$ and $\mathsf{st}(\omega_i^\bullet) = \omega_i^+$:

$$\omega_1^+ = \omega_2^+ = \frac{b_2(\omega_2^-)\,\omega_1^- + b_1(\omega_1^-)\,\omega_2^-}{b_1(\omega_1^-) + b_2(\omega_2^-)}, \tag{17}$$

where we recall that $\mathsf{st}(\omega)$ is the standard part of ω, see the beginning of Section 5.2. Eq. (17) provides us with the reset values for the positions in the engaged mode, which is enough to restart the simulation in this mode.

Figure 7 shows a simulation of the clutch where the resets are computed following this approach. As expected, the reset value sits between the two values of ω_1^- and ω_2^- when $\gamma : \mathrm{F} \to \mathrm{T}$ (at $t = 5$s), and the transition is continuous at the second reset (at $t = 10$s). An alternative approach for the computation of the reset values, which does not require the elimination of impulsive variables, is developed in (Benveniste, Caillaud, and Malandain 2020), see also (Benveniste, Caillaud, and Malandain 2021).

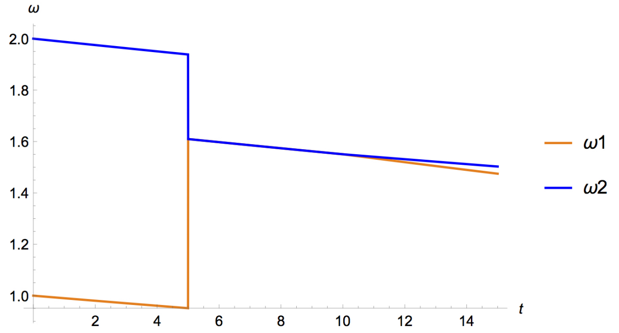


Figure 7. Simulation of the clutch model with resets. Mode change $\mathrm{F} \to \mathrm{T}$ occurs at $t = 5s$ and mode change $\mathrm{T} \to \mathrm{F}$ occurs at $t = 10s$.

6 The Cup-and-Ball example

Using (3), the original model (2) is rewritten as

$$\left\{\begin{array}{rll} & 0 = x'' + \lambda x & (e_1) \\ & 0 = y'' + \lambda y + g & (e_2) \\ & \gamma = [s \leq 0] & (k_0) \\ \texttt{if } \gamma \texttt{ do} & 0 = L^2 - (x^2 + y^2) & (k_1) \\ \texttt{and} & 0 = \lambda + s & (k_2) \\ \texttt{if not } \gamma \texttt{ do} & 0 = \lambda & (k_3) \\ \texttt{and} & 0 = (L^2 - (x^2 + y^2)) - s & (k_4) \end{array}\right. \quad (18)$$

As stated in Section 2.2, two issues have to be addressed by our structural analysis: the expected impulsive behavior of the accelerations at mode changes, and the insufficient specification of the nature (elastic, inelastic or in between) of the impact.

We implicitly add to model (18) the following two equations, for each state variable v:

$$v' = \frac{v^{\bullet} - v}{\partial} \;\; ; \;\; v'' = \frac{v^{\bullet 2} - 2v^{\bullet} + v}{\partial^2} \,, \quad (19)$$

where

$$\begin{array}{rcl} v^{\bullet}(t) & =_{\text{def}} & v(t + \partial)\,, \\ v^{\bullet 2}(t) & =_{\text{def}} & v(t + 2\partial) \text{ and, more generally,} \\ v^{\bullet n}(t) & =_{\text{def}} & v(t + n\partial)\,. \end{array}$$

Equation (19) means that the derivatives x', y', x'', y'' are interpreted using the explicit first-order Euler scheme with an *infinitesimal time step* ∂. Note that (19) implies

$$x'' \;\; = \;\; \frac{x'^{\bullet} - x'}{\partial} \;\; . \quad (20)$$

After performing the substitutions given by (19), we observe that the subsystem collecting equations (k_0)–(k_4) is a logico-numerical fixpoint equation, with dependent variables $x^{\bullet 2}, y^{\bullet 2}, \lambda, \gamma$. A possible solution would consist in performing a relaxation, by iteratively updating the numerical variables based on the previous value for the guards, and then re-evaluating the guard based on the updated values of the numerical variables, hoping for a fixpoint to occur. Such fixpoint equation, however, can have zero, one, several, or infinitely many solutions. No characterization exists that could serve as a basis for a (graph-based) structural analysis. *We thus decide to refuse solving such mixed logico-numerical systems.*

As a consequence, we are unable to evaluate guard γ, so that the mode the system is in cannot be determined: model (18) is rejected.

To break the fixpoint equation defining γ, we choose to systematically introduce infinitesimal delays to guards. For the Cup-and-Ball, the predicate $s \leq 0$ then defines the value of the guard *at the next nonstandard instant.*[4] This yields the corrected model (21), where the modification is highlighted in red.

$$\left\{\begin{array}{rll} & 0 = x'' + \lambda x & (e_1) \\ & 0 = y'' + \lambda y + g & (e_2) \\ & \gamma^{\bullet} = [s \leq 0]; \gamma(0) = \mathrm{F} & (k_0) \\ \texttt{if } \gamma \texttt{ do} & 0 = L^2 - (x^2 + y^2) & (k_1) \\ \texttt{and} & 0 = \lambda + s & (k_2) \\ \texttt{if not } \gamma \texttt{ do} & 0 = \lambda & (k_3) \\ \texttt{and} & 0 = (L^2 - (x^2 + y^2)) - s & (k_4) \end{array}\right. \quad (21)$$

This model is understood in the nonstandard setting, meaning that the derivatives are expanded using (19). The leading variables in all modes are $\lambda, s, x^{\bullet 2}, y^{\bullet 2}$.

6.1 Structural analysis of mode change γ:F→T

Due to equation (k_1), the mode $\gamma = \mathrm{T}$ (where the rope is straight) requires index reduction. We thus augment model (21) with the two latent equations shown in red:

$$\left\{\begin{array}{rll} & 0 = x'' + \lambda x & (e_1) \\ & 0 = y'' + \lambda y + g & (e_2) \\ & \gamma^{\bullet} = [s \leq 0]; \gamma(0) = \mathrm{F} & (k_0) \\ \texttt{if } \gamma \texttt{ do} & 0 = L^2 - (x^2 + y^2) & (k_1) \\ \texttt{and} & 0 = L^2 - (x^2 + y^2)^{\bullet} & (k_1^{\bullet}) \\ \texttt{and} & 0 = L^2 - (x^2 + y^2)^{\bullet 2} & (k_1^{\bullet 2}) \\ \texttt{and} & 0 = \lambda + s & (k_2) \\ \texttt{if not } \gamma \texttt{ do} & 0 = \lambda & (k_3) \\ \texttt{and} & 0 = (L^2 - (x^2 + y^2)) - s & (k_4) \end{array}\right. \quad (22)$$

Note that, as in System 10, the two latent equations $(k_1^{\bullet})$ and $(k_1^{\bullet 2})$ were obtained by *shifting* (k_1) *forward*, which is equivalent to differentiating it for the structural analysis. To perform structural analysis at the considered mode change, we unfold model (22) at the successive instants

$${}^{\bullet 2}t \;=_{\text{def}}\; t - 2\partial, \quad {}^{\bullet}t \;=_{\text{def}}\; t - \partial, \quad \text{and} \;\; t\,,$$

where t denotes the current instant. In the following, equation (e_1) at the instant $t - 2\partial$ (respectively, $t - \partial$) will be denoted by $({}^{\bullet 2}e_1)$ (resp., $({}^{\bullet}e_1)$).

[4] The condition triggering the mode change is based on the positions, which remain continuous at mode changes, even though the velocities are discontinuous. As a result, the shifting of this guard by an infinitesimal time step only yields an infinitesimal change in the values of state variables, which will be erased by the standardization process, so that the numerical solution is not impacted by this change in the model.

In this unfolding, the two equations (k_1) and $(k_1^{\bullet})$ are in conflict with selected equations from the previous two instants, shown in blue in the following subsystem, whose dependent variables are the leading variables at instants $t-2\partial$ and $t-\partial$, namely $x, y, {}^{\bullet 2}\lambda; x^{\bullet}, y^{\bullet}, {}^{\bullet}\lambda$:

$$\left\{\begin{array}{ll}
0 = \frac{x-2\,{}^{\bullet}x+{}^{\bullet 2}x}{\partial^2} + {}^{\bullet 2}\lambda\,{}^{\bullet 2}x & ({}^{\bullet 2}e_1) \\
0 = \frac{y-2\,{}^{\bullet}y+{}^{\bullet 2}y}{\partial^2} + {}^{\bullet 2}\lambda\,{}^{\bullet 2}y + g & ({}^{\bullet 2}e_2) \\
0 = \frac{x^{\bullet}-2x+{}^{\bullet}x}{\partial^2} + {}^{\bullet}\lambda\,{}^{\bullet}x & ({}^{\bullet}e_1) \\
0 = \frac{y^{\bullet}-2y+{}^{\bullet}y}{\partial^2} + {}^{\bullet}\lambda\,{}^{\bullet}y + g & ({}^{\bullet}e_2) \\
0 = L^2-(x^2+y^2) & (k_1) \\
0 = L^2-(x^2+y^2)^{\bullet} & (k_1^{\bullet})
\end{array}\right.$$

We resolve this conflict by applying causality Principle 1, which leads to erasing, in model (22), equations (k_1) and $(k_1^{\bullet})$ at the instant of mode change ${}^{\bullet}\gamma{=}\mathrm{F}, \gamma{=}\mathrm{T}$. This yields:

$$\text{at}\left[\begin{array}{c} {}^{\bullet}\gamma{=}\mathrm{F} \\ \gamma{=}\mathrm{T} \end{array}\right]:\left\{\begin{array}{ll}
0 = x'' + \lambda x & (e_1) \\
0 = y'' + \lambda y + g & (e_2) \\
0 = L^2-(x^2+y^2)^{\bullet 2} & (k_1^{\bullet 2}) \\
0 = \lambda + s & (k_2)
\end{array}\right. \tag{23}$$

System (23) uniquely determines all the leading variables from the state variables x, y and $x^{\bullet}, y^{\bullet}$. In turn, equations (k_1) and $(k_1^{\bullet})$, which were erased from this model, are not satisfied. At the next instant, i.e., when ${}^{\bullet 2}\gamma{=}\mathrm{F}, {}^{\bullet}\gamma{=}\mathrm{T}, \gamma{=}\mathrm{T}$, the same argument is used. We thus erase, in model (22), the only equation (k_1) at the next instant. This yields:

$$\text{at}\left[\begin{array}{c} {}^{\bullet 2}\gamma{=}\mathrm{F} \\ {}^{\bullet}\gamma{=}\mathrm{T} \\ \gamma{=}\mathrm{T} \end{array}\right]:\left\{\begin{array}{ll}
0 = x'' + \lambda x & (e_1) \\
0 = y'' + \lambda y + g & (e_2) \\
0 = L^2-(x^2+y^2)^{\bullet} & (k_1^{\bullet}) \\
0 = L^2-(x^2+y^2)^{\bullet 2} & (k_1^{\bullet 2}) \\
0 = \lambda + s & (k_2)
\end{array}\right. \tag{24}$$

Note that $(k_1^{\bullet})$ is a consistency equation that is satisfied by the state variables $x^{\bullet}, y^{\bullet}$. In turn, equation (k_1), which was erased from this model, is not satisfied. At subsequent instants, equation erasure is no longer needed.

This completes the nonstandard structural analysis of the mode change $\gamma:\mathrm{F}{\to}\mathrm{T}$, i.e., when the rope gets straight.

6.2 Getting effective code for restart

Code generation for restarts consists in standardizing nonstandard systems (23) and (24), in a way similar to Section 5.4. We focus on the standardization of the mode change $\gamma:\mathrm{F}\to\mathrm{T}$, i.e., when the rope gets straight. Our task is to standardize systems (23) and (24), by targeting discrete-time dynamics, for the two successive instants composing the restart phase. This will provide us with restart values for positions and velocities.

Due to the expansion of derivatives in equations $(e_1, e_2, e_1^{\bullet}, e_2^{\bullet})$, tensions λ and $\lambda^{\bullet}$ are both impulsive, hence so are s and $s^{\bullet}$ by $(k_2, k_2^{\bullet})$. We eliminate the impulsive variables by ignoring $(k_2, k_2^{\bullet})$, combining (e_1) and (e_2) to eliminate λ, and $(e_1^{\bullet})$ and $(e_2^{\bullet})$ to eliminate $\lambda^{\bullet}$. This yields:

$$\text{at}\left[\begin{array}{c} {}^{\bullet}\gamma{=}\mathrm{F} \\ \gamma{=}\mathrm{T} \end{array}\right]:\left\{\begin{array}{l}
0 = y''x + gx - x''y \\
0 = L^2-(x^2+y^2)^{\bullet 2}
\end{array}\right. \tag{25}$$

$$\text{at}\left[\begin{array}{c} {}^{\bullet 2}\gamma{=}\mathrm{F} \\ {}^{\bullet}\gamma{=}\mathrm{T} \\ \gamma{=}\mathrm{T} \end{array}\right]:\left\{\begin{array}{l}
0 = y''x + gx - x''y \\
0 = L^2-(x^2+y^2)^{\bullet} \\
0 = L^2-(x^2+y^2)^{\bullet 2}
\end{array}\right. \tag{26}$$

In System (25) we expand second derivatives using (19), whereas in System (26) we expand them using (20). Consequently, System (25) has dependent variables $x^{\bullet 2}, y^{\bullet 2}$, whereas System (26) has dependent variables $x'^{\bullet}, y'^{\bullet}$. We are now ready to standardize the two systems.

System (25) to define restart positions: We expand second derivatives using (19):

$$\left\{\begin{array}{l}
0 = (y^{\bullet 2}-2y^{\bullet}+y)x - (x^{\bullet 2}-2x^{\bullet}+x)y + \partial^2 gx \\
0 = L^2-(x^2+y^2)^{\bullet 2}
\end{array}\right. \tag{27}$$

Setting $\partial = 0$ in this system yields a structurally regular system. Hence, by a theorem proved in (Benveniste, Caillaud, and Malandain 2020), the so obtained system is the correct standardization of System (27). In contrast, had we set $\partial = 0$ in System (23) (without eliminating impulsive variable λ), we would get a structurally singular system, an incorrect standardization.

In System (27) with $\partial = 0$, we can interpret x and $x^{\bullet}$ as the left-limit x^- of state variable x in previous mode, and $x^{\bullet 2}$ as the restart value x^+ for the new mode. This yields

$$\left\{\begin{array}{l}
0 = (y^+-y^-)x^- - (x^+-x^-)y^- \\
0 = L^2-(x^2+y^2)^+
\end{array}\right. \tag{28}$$

which determines the restart values for positions. The constraint that the rope is straight is satisfied. Furthermore, as $0 = L^2-(x^2+y^2)^-$ also holds (the rope is straight at the mode change), $x^+ = x^-, y^+ = y^-$ is the unique solution of (28): positions are continuous.

System (26) to define restart velocities: We expand second derivatives using (20):

$$\left\{\begin{array}{l}
0 = (y'^{\bullet}-y')x - (x'^{\bullet}-x')y + \partial.gx \\
0 = L^2-(x^2+y^2)^{\bullet} \\
0 = L^2-(x^2+y^2)^{\bullet 2}
\end{array}\right. \tag{29}$$

By expanding $x^{\bullet 2} = x^{\bullet} + \partial x'^{\bullet}$, the right-hand side of the last equation rewrites

$$\begin{array}{rcl}
L^2-(x^2+y^2)^{\bullet 2} & = & L^2-(x^2+y^2)^{\bullet} \\
& + & 2\partial(x^{\bullet}x'^{\bullet}+y^{\bullet}y'^{\bullet}) \\
& + & \partial^2\big((x'^{\bullet})^2+(y'^{\bullet})^2\big) \\
& = & 0 \;\; (\text{using (29)}) \\
& + & 2\partial(x^{\bullet}x'^{\bullet}+y^{\bullet}y'^{\bullet}) + O(\partial^2)
\end{array} \tag{30}$$

Using this expansion, setting $\partial = 0$ in (29) yields

$$\left\{\begin{array}{l}
0 = (y'^{\bullet}-y')x - (x'^{\bullet}-x')y \\
0 = x^{\bullet}x'^{\bullet} + y^{\bullet}y'^{\bullet}
\end{array}\right. \tag{31}$$

where the dependent variables are now $x'^{\bullet}, y'^{\bullet}$—other variables are state variables whose values were set at previous time steps. System (31) is structurally regular, hence, it is the correct standardization of System (29).

To get effective code for restart, we perform, in (31), the following substitutions, where superscripts $^-$ and $^+$ denote left- and right-limits, and continuity of positions is used:

$$x = x^- \ ; \ x^\bullet = x^+ \quad \text{and} \quad x' = x'^- \ ; \ x'^+ = x'^\bullet \tag{32}$$

and similarly for y. This finally yields

$$\begin{cases} 0 = (y'^+ - y'^-)x^- - (x'^+ - x'^-)y^- \\ 0 = x^+ x'^+ + y^+ y'^+ \end{cases} \tag{33}$$

System (33) determines x'^+ and y'^+, which are the velocities for restart. The second equation guarantees that the velocity will be tangent to the constraint. With (28) and (33), we determine the restart conditions for positions and velocities. Invariants from the physics are satisfied.

Our reasoning so far produces a behavior in which the two modes (free motion and straight rope) gently alternate; the system always stays in one mode for some positive period of time before switching to the other mode.

This indeed amounts to assuming that the impact is totally inelastic at mode change, an assumption that was not explicit at all in (21)***.*** So, what happened? In fact, ***the straight rope mode was implicitly assumed to last for at least three nonstandard successive instants, since we allowed ourselves to shift*** (k_1) ***twice.***

6.3 Handling transient modes

Let us instead assume elastic impact, represented by the cascade of mode changes $\gamma : \text{F} \rightarrow \text{T} \rightarrow \text{F}$, reflecting that the straight rope mode is *transient* (it is left immediately after being reached).

Consider again model (21). We regard the instant of the cascade when $\gamma = \text{T}$ occurs as the current instant. We cannot add latent equations by simply shifting (k_1), since these shifted versions are not active in the mode $\gamma = \text{F}$. Set

$$\begin{aligned} S(\text{T}) &= \{(e_1), (e_2), (k_1), (k_2)\} \\ S(\text{F}) &= \{(e_1), (e_2), (k_3), (k_4)\} \end{aligned}$$

Systems $S^\bullet(\text{T})$ and $S^\bullet(\text{F})$ are obtained by shifting once the equations constituting $S(\text{T})$ and $S(\text{F})$; systems $S^{\bullet k}(\text{T})$ and $S^{\bullet k}(\text{F})$ are defined similarly for all $k \in \mathbb{N}$. Consider the *differentiation array* originally proposed by (Campbell and Gear 1995), except that we take into account the trajectory $\text{T}, \text{F}, \text{F}, \dots$ for guard γ. Using shifting instead of differentiation yields the following *difference array*:

$$\mathscr{A}_n(S) =_{\text{def}} \left[\ S(\text{T}) \quad S^\bullet(\text{F}) \quad S^{\bullet 2}(\text{F}) \quad \dots \quad S^{\bullet n}(\text{F}) \ \right]^T \tag{34}$$

The dependent variables of System $\mathscr{A}_n = 0$ are $x^{\bullet 2}, y^{\bullet 2}, \lambda$, whereas $x^{\bullet(k+2)}, y^{\bullet(k+2)}, \lambda^{\bullet(k)}, k > 0$ must be eliminated. We look for the smallest n such that $\mathscr{A}_n = 0$ is structurally nonsingular in this sense. Unfortunately, although shifting (k_4) twice in System (21) produces one more equation involving the leading variables $x^{\bullet 2}, y^{\bullet 2}$, this equation also involves the new variable $s^{\bullet 2}$, which keeps the augmented system underdetermined; shifting other equations fails as well. Therefore, the structural analysis rejects this model as being underdetermined at transient mode $\gamma = \text{T}$.

The user is then asked to provide one more equation. For example, they could specify an impact law for the velocity y' by providing the equation $(y')^+ = -(1-\alpha)(y')^-$, where $0 \leq \alpha < 1$ is a fixed damping coefficient. This is reinterpreted in the nonstandard domain as $y'^\bullet = -(1-\alpha)y'$, yielding the following refined system for use at mode γ=T within the cascade γ:F→T→F:

$$\begin{cases} 0 = x'' + \lambda x & (e_1) \\ 0 = y'' + \lambda y + g & (e_2) \\ 0 = y'^\bullet + (1-\alpha)y' & (\tau_1) \\ 0 = L^2 - (x^2 + y^2) & (k_1) \\ 0 = \lambda + s & (k_2) \end{cases} \tag{35}$$

The modified difference array is now structurally nonsingular. The so modified model is accepted and two-step restart code for the mode change is generated as before.

6.4 Consequences for the modeling language

Through the Cup-and-Ball example, *we demonstrated the need for the following user-given information: is the current mode long or transient?* Long / Transient is an information regarding modes, that cannot be found by an automatic inspection of the model. It must be inferred from understanding the system physics and must be manually specified. The natural way of performing this is to provide a different syntax for specifying long modes on the one hand, and events corresponding to transient modes on the other hand (mode changes separating two successive long modes need not be specified).

The '`if`' and '`when`' statements of the Modelica language are fit candidates for this purpose. We devote the '`if`' statement to long-lasting modes specified by a predicate, while the '`when`' statement, pointing to the event when a predicate switches from F to T, could be further restricted to be a zero-crossing condition, by which a $\mathbb{R}$-valued expression crosses zero from below (Bourke and Pouzet 2013). Using this feature, the Cup-and-Ball example with elastic impact is specified as follows:

$$\left\{ \begin{array}{rll} & 0 = x'' + \lambda x & (e_1) \\ & 0 = y'' + \lambda y + g & (e_2) \\ & \gamma = [s^- \leq 0]; \gamma(0) = \text{F} & (k_0) \\ \texttt{when}\ \gamma\ \texttt{do} & y'^+ = -\alpha y'^- & (\tau_1) \\ \texttt{if not}\ \gamma\ \texttt{do} & 0 = \lambda & (k_3) \\ \texttt{and} & 0 = (L^2 - (x^2 + y^2)) - s & (k_4) \end{array} \right.$$

7 Mechanization of the process

The approach developed in Sections 5.4 and 6.2 is a systematic way to define the solution of a multimode DAE system, than can be generalized to large-scale and/or multi-physics models. However, this reasoning:

- *Requires identifying impulsive variables.* We present in companion paper (Benveniste, Caillaud, and Malandain 2021) a calculus for this, which is ready for automatization in a tool (this is under development in our IsamDAE tool[5]).

- *Requires eliminating impulsive variables.* This is easy if impulsive variables enter linearly in the model—this was the case for the clutch and the cup-and-ball examples. It is highly costly but still doable if impulsive variables enter polynomially in the model. It cannot be done practically in other cases.

- *Relies on a clever choice of how to map nonstandard variables to restart conditions.* This was straightforward for the clutch, but definitely not for the cup-and-ball (Section 6.2), where expansion (19) for the derivatives was used for resetting positions, whereas expansion (20) was used for resetting velocities.

This is not realistic for implementation in a tool, except for restricted classes of systems in which impulsive variables enter linearly in the model.

However, in companion paper (Benveniste, Caillaud, and Malandain 2021) we propose an alternative, which is a good candidate for implementation, based on an *impulse analysis*. This post-processing of the structural analysis at mode changes is a simple and systematic calculus that identifies impulsive variables at compile time and quantifies their (possibly infinite) magnitude order, called *impulse order.* When finite, the impulse order can be used to rescale impulsive variables, which allows for computing restart values for state variables as well as rescaled impulsive variables. When impulse orders are infinite, rescaling no longer applies. It is, however, still possible to compute restart conditions by using the nonstandard equations with a small positive (standard) time step. This provides converging approximations for the non-impulsive variables (the state variables in particular).

8 Conclusion

Through the case study of two examples of multimode DAEs that are currently not handled by the existing Modelica tools (with the notable exception of ModiaMath), we presented a mathematically sound and physics-agnostic compilation process for DAE-based physical systems modeling languages. This method relies, in particular, on an extension of structural analysis to multimode systems, that allows the handling of both modes and mode changes in a unified framework.

Both examples studied in this paper are multimode models with mode-dependent differentiation index and impulsive behaviour at mode changes, which is not well supported by existing Modelica compilers. This paper showed how our approach handles such models: not only is the structural analysis correctly performed in all continuous modes, but the computation of restart values at mode changes is also handled at compile time, unless an under/over-determination at a mode change event causes the model to be rejected with proper diagnostics.

Ongoing works include the effective mechanization of the process, which is detailed in the companion paper (Benveniste, Caillaud, and Malandain 2021). An important bottleneck of this approach is that it needs to handle all modes and all possible mode changes at compile time: unfortunately, the number of modes tends to be roughly exponential in the size of the model, and the *a priori* number of mode changes is at least proportional to the square of the number of modes. This is a limitation of a model representation in which one characterizes the subset of equations and variables active in any given mode.

A possible way of alleviating this issue is by shifting to a dual representation, that provides predicates characterizing the set of modes in which each equation and each variable is active. In practice, not only does this approach lead to a much more compact representation, but it also allows for the design of efficient structural analysis methods for multimode DAE systems, working in an 'all-modes-at-once' fashion. Such a method was implemented in the IsamDAE tool, and first results are reported in (Caillaud, Malandain, and Thibault 2020). The examples coming with this tool already include thermodynamical, electrical and pneumatic models. Although only the structural analysis of long modes is currently performed, the implementation of the structural analysis of mode changes is in progress.

Acknowledgements

The authors are indebted to Hilding Elmqvist and Martin Otter, John Pryce, and Vincent Acary. Khalil Ghorbal participated to the first version of this approach.

This work was supported by the FUI ModeliScale DOS0066450/00 French national grant (2018-2021) and the Inria IPL ModeliScale large scale initiative (2017-2021, `https://team.inria.fr/modeliscale/`). Dymola licences were provided to the authors by Dassault Systèmes in the context of the FUI ModeliScale project.

References

Benveniste, Albert, Benoit Caillaud, Hilding Elmqvist, et al. (2017-04). "Structural Analysis of Multi-Mode DAE Systems". In: *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, pp. 253–263. ISBN: 978-1-4503-4590-3. DOI: 10.1145/3049797.3049806.

Benveniste, Albert, Benoit Caillaud, Hilding Elmqvist, et al. (2019). "Multi-Mode DAE Models - Challenges, Theory and Implementation". In: *Computing and Software Science - State of the Art and Perspectives*. Ed. by Bernhard Steffen and Gerhard J. Woeginger. Vol. 10000. Lecture Notes in Computer

[5] `https://allgo18.inria.fr/apps/isamdae`

Science. Springer, pp. 283–310. ISBN: 978-3-319-91907-2. DOI: 10.1007/978-3-319-91908-9_16.

Benveniste, Albert, Benoit Caillaud, and Mathias Malandain (2020). "The mathematical foundations of physical systems modeling languages". In: *Annual Reviews in Control* 50, pp. 72–118. ISSN: 1367-5788. DOI: 10.1016/j.arcontrol.2020.08.001.

Benveniste, Albert, Benoit Caillaud, and Mathias Malandain (2021-09). "Compile Time Impulse Analysis in Modelica". In: *Proceedings of the 14th International Modelica Conference*. Linköping University Electronic Press.

Bourke, Timothy and Marc Pouzet (2013-04). "Zélus: A Synchronous Language with ODEs". In: *Hybrid Systems: Computation and Control (HSCC)*. ACM. Philadelphia, USA, pp. 113–118.

Caillaud, Benoit, Mathias Malandain, and Joan Thibault (2020-04). "Implicit Structural Analysis of Multimode DAE Systems". In: *23rd ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2020)*. Sydney, Australia. DOI: 10.1145/3365365.3382201.

Campbell, Stephen L. and C. William Gear (1995). "The index of general nonlinear DAEs". In: *Numer. Math.* 72, pp. 173–196.

Dassault Systèmes AB (2020). *Dymola official webpage*. Accessed: 2021-06-28. URL: https://www.3ds.com/products-services/catia/products/dymola/.

Dulmage, Andrew L. and Nathan S. Mendelsohn (1958). "Coverings of Bipartite Graphs". In: *Canadian Journal of Mathematics* 10, pp. 517–534. DOI: 10.4153/CJM-1958-052-0.

Elmqvist, Hilding, Fabien Gaucher, et al. (2012-09). "State Machines in Modelica". In: *Proc. of the Int. Modelica Conference*. Ed. by Martin Otter and Dirk Zimmer. Modelica Association. Munich, Germany, pp. 37–46.

Elmqvist, Hilding, Sven Erik Mattsson, and Martin Otter (2014-09). "Modelica extensions for multi-mode DAE systems". In: *Proc. of the 10th Int. Modelica Conference*. Ed. by Hubertus Tummescheit and Karl-Erik Arzen. Modelica Association. Lund, Sweden.

Fritzson, Peter et al. (2020). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Höger, Christoph (2014). "Dynamic structural analysis for DAEs". In: *Proceedings of the 2014 Summer Simulation Multiconference, SummerSim 2014, Monterey, CA, USA, July 6-10, 2014*. SCS/ ACM, p. 12.

Höger, Christoph (2017). "Elaborate control: variable-structure modeling from an operational perspective". In: *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, EOOLT '17, Weßling, Germany, December 1, 2017*. Ed. by Dirk Zimmer and Bernhard Bachmann. ACM, pp. 51–60. ISBN: 978-1-4503-6373-0. DOI: 10.1145/3158191.3158198.

Liberzon, Daniel and Stephan Trenn (2012). "Switched nonlinear differential algebraic equations: Solution theory, Lyapunov functions, and stability". In: *Automatica* 48.5, pp. 954–963. DOI: 10.1016/j.automatica.2012.02.041.

Lindstrøm, Tom (1988). "An Invitation to Nonstandard Analysis". In: *Nonstandard Analysis and its Applications*. Ed. by N.J. Cutland. Cambridge Univ. Press, pp. 1–105.

Mattsson, Sven Erik, Martin Otter, and Hilding Elmqvist (1999). "Modelica Hybrid Modeling and Efficient Simulation". In: *38th IEEE Conference on Decision and Control*. Ed. by IEEE, pp. 3502–3507.

Mattsson, Sven Erik, Martin Otter, and Hilding Elmqvist (2015-09). "Multi-Mode DAE Systems with Varying Index". In: *Proc. of the 11th Int. Modelica Conference*. Ed. by Hilding Elmqvist and Peter Fritzson. Modelica Association. Versailles, France.

Mattsson, Sven Erik and Gustaf Soderlind (1993). "Index Reduction in Differential-Algebraic Equations Using Dummy Derivatives". In: *SIAM Journal on Scientific Computing* 14.3, pp. 677–692. DOI: 10.1137/0914043.

Nilsson, Henrik and George Giorgidze (2010). "Exploiting structural dynamism in Functional Hybrid Modelling for simulation of ideal diodes". In: *Czech Technical University Publishing House*.

Pantelides, Constantinos C. (1988). "The consistent initialization of differential-algebraic systems". In: *SIAM J. Sci. Stat. Comput.* 9.2, pp. 213–231.

Pepper, Peter et al. (2011). "A Compositional Semantics for Modelica-style Variable-structure Modeling". In: *4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*.

Pryce, John D. (2001). "A simple structural analysis method for DAEs". In: *BIT* 41.2, pp. 364–394.

Robinson, Abraham (1996). *Nonstandard Analysis*. Princeton Landmarks in Mathematics. ISBN: 0-691-04490-2.

Trenn, Stephan (2009a). "Distributional Differential Algebraic Equations". PhD thesis. Technischen Universität Ilmenau.

Trenn, Stephan (2009b). "Regularity of distributional differential algebraic equations". In: *MCSS* 21.3, pp. 229–264. DOI: 10.1007/s00498-009-0045-4.

A Reduced Index Mode-Independent Structure Model Transformation for Multimode Modelica Models

Benoît Caillaud[1] Mathias Malandain[1] Albert Benveniste[1]

[1]Inria Centre de Rennes Bretagne Atlantique, University of Rennes 1, France,
{benoit.caillaud,mathias.malandain,albert.benveniste}@inria.fr

Abstract

Since its 3.3 release, Modelica offers the possibility to specify models of dynamical systems with multiple modes having different DAE-based dynamics. However, the handling of such models by the current Modelica tools is not satisfactory, with mathematically sound models yielding exceptions at runtime. In this article, we propose a systematic way of rewriting a multimode Modelica model, based on the results of an already implemented multimode structural analysis. The rewritten Modelica model is guaranteed to be correctly compiled by state-of-the-art Modelica tools. Simulation results are presented on a simple, yet meaningful, physical system whose original Modelica model is not correctly handled by state-of-the-art Modelica tools.

Keywords: Modelica, multimode DAE, structural analysis, model transformations

1 Introduction

Since version 3.3, the Modelica language offers the possibility of specifying *multimode dynamics,* by describing state machines with different DAE dynamics in each different state (Elmqvist et al. 2012). This feature enables describing large complex cyber-physical systems with different behaviors in different modes.

While being undoubtedly valuable, multimode modeling has been the source of serious difficulties for non-expert users of the current generation of Modelica tools. Indeed, while many large-scale Modelica models are properly handled, some physically meaningful models do not result in correct simulations with most Modelica tools. It is actually not difficult to construct such problematic models, thus, chances are significant to produce such bad cases in large models. Quite often, end users have to ask Modelica experts, or even tool developers themselves, to tweak their models in order to make them work as expected. This situation hinders a wider spreading of Modelica tools among a larger class of users, such as Simulink-trained engineers.

New language constructs have been proposed in the past to address the limited capability of the Modelica language to handle multimode models. The Sol (Zimmer 2010) and the Hydra (Giorgidze and Nilsson 2011; Nilsson and Giorgidze 2010) languages have been designed with the capability to enable and disable equations, depending on the current mode of the system. For both languages, structural analysis is performed at runtime, when the system switches to a new mode.

Some years ago, we started a project aiming at addressing all the above issues, with a different perspective in mind, that consists in privileging compile-time, rather than runtime, analyses. In (Benveniste, Caillaud, Elmqvist, et al. 2019; Benveniste, Caillaud, and Malandain 2020) we explain our approach, and we illustrate it on two simple, yet physically meaningful, examples in (Benveniste, Caillaud, and Malandain 2021). One key feature of this approach is structural analysis: it is important that this task is performed for each mode and each mode change at compile time, in order to avoid unexpected behaviour at runtime. In (Caillaud, Malandain, and Thibault 2020), we present an effective approach to achieve compile-time, mode-dependent, structural analysis *without enumerating the modes* (as this would not be able to scale up). The advantages we see in our approach are twofold: (i) it provides, at compile-time, invaluable information that helps users debug their models, and (ii) efficient code generation is possible since the automatic differentiation of latent equations can be done at compile-time and blocks of equations can be compiled into functions that can be passed directly to numerical solvers, without any further processing.

In this article, we demonstrate how the results of this multimode structural analysis can be used for transforming a multimode Modelica model into its RIMIS (Reduced Index Mode-Independent Structure) form, which is guaranteed to yield correct execution on state-of-the-art Modelica tools. This method is illustrated on a water tank model for which current Modelica tools fail to execute; in this model, the differentiation index depends on the mode, which is a problem for these tools. In particular, we explain how existing structural analysis methods fail to yield correct execution code for this model, then demonstrate the generation of a target code under RIMIS form, resulting in a correct simulation of the model. Our approach is then formalized for its broad application to problematic multimode models.

DOI
10.3384/ecp21181519

```
model WaterTank
  Real t(start=0,fixed=true); // time (to define input flow)
  constant Real xmax = 1.0; // max water quantity
  constant Real xmin = 0.0; // min water quatity
  constant Real y0 = 6.667; // default output flow
  constant Real rho = 0.8; // input flow parameter
  Real x(start=0.5,fixed=true); // stored water mass
  Real yh; // output flow correction, when tank is full
  Real yl; // output flow correction, when tank is empty
  Real z; // input flow
  Real sh; // parameter of the full-tank CC
  Real sl; // parameter of the empty-tank CC
  Boolean bh(start=false,fixed=true); // mode full-tank
  Boolean bl(start=false,fixed=true); // mode empty-tank
  // bh and bl satisfy assertion not (bh and bl)
equation
  // input flow law
  /* et: */ der(t)=1;
  /* e1: */ z = rho*y0*(1+
            Modelica.Math.cos(2*Modelica.Constants.pi*t));
  // tank level differential equation
  /* e2: */ der(x) = z + yl - yh - y0;
  // Complementarity condition 0 <= xmax - x # yh >= 0
  bh = (sh >= 0);
  /* eh1: */ sh = if bh then yh else x - xmax;
  /* eh2: */ 0 = if bh then x - xmax else yh;
  // complementarity condition 0 <= x - xmin # yl >= 0
  bl = (sl >= 0);
  /* el1: */ sl = if bl then yl else xmin - x;
  /* el2: */ 0 = if bl then xmin - x else yl;
end WaterTank;
```

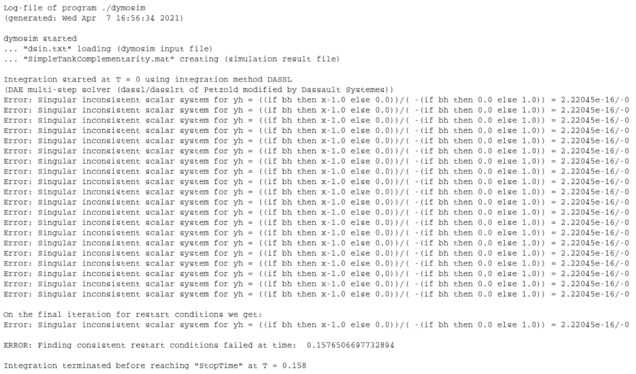
Figure 1. Modelica model of the Water Tank system. Comments of the form `/* id: */` define equation labels appearing in the dependency graphs in Figures 3 and 4.

2 The Water Tank system and failed simulations with Modelica tools

The Water Tank system is a simple model of a closed tank with a variable water inflow `z` and a default outflow `y0`, where water is considered incompressible. When the tank is full, a positive flow correction `yh` is added to the outflow, as the tank cannot store more water; conversely, when the tank is empty, a negative flow correction `yl` is added to the outflow.

The corresponding Modelica model, given in Figure 1, uses two complementarity conditions (Van Der Schaft and Schumacher 1998) for the flow corrections. The first one, encoded by the multimode equations `eh1` and `eh2`, depends on the Boolean variable `bh`, which is true if and only if variable `sh` is nonnegative. The combined effect of these two equations is that `xmax` − `x` and `yh` are always nonnegative, and that at least one of those is equal to 0 at any time. Equations `el1` and `el2` encode the second complementarity condition in a similar way.

This model fails to simulate properly with both OpenModelica 1.17.0 (Fritzson et al. 2020) and Dymola 2021 (Dassault Systèmes AB 2020); Figure 2 shows the output of Dymola 2021. The root cause is that state-of-the-art Modelica tools perform an approximate structural analysis, disregarding the fact that the structure of the system is mode-dependent. A more detailed explanation is provided in Section 3.1.



```
Log-file of program ./dymosim
(generated: Wed Apr  7 16:56:34 2021)

dymosim started
... "dsin.txt" loading (dymosim input file)
... "SimpleTankComplementarity.mat" creating (simulation result file)

Integration started at T = 0 using integration method DASSL
(DAE multi-step solver (dassl/dasslrt of Petzold modified by Dassault Systemes))
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0

On the final iteration for restart conditions we get:
Error: Singular inconsistent scalar system for yh = ((if bh then x-1.0 else 0.0))/( -(if bh then 0.0 else 1.0)) = 2.22045e-16/-0

ERROR: Finding consistent restart conditions failed at time:  0.1576506697732894

Integration terminated before reaching "StopTime" at T = 0.158
```

Figure 2. Simulation of the Water Tank system with Dymola 2021, failing with a division by zero exception.

3 Structural analysis: from single- to multi-mode

DAE-based languages and tools rely on *structural analysis* as a required preprocessing step of a DAE system, needed for the generation of simulation code. This analysis turns the original system into a *reduced index* (Campbell and Gear 1995) system, amenable to numerical solvers, by differentiating one or several times all or part of the equations.

Well-understood methods such as the renowned Pantelides algorithm (Pantelides 1988), the dummy derivatives method (Mattsson and Soderlind 1993) or the less known Σ-method (Pryce 2001) can be used for single-mode DAE systems; however, the structural analysis of multimode DAE systems is still in its infancy, and even state-of-the-art Modelica tools have to rely, at least in part, on an approximate 'single-mode' structural analysis for the generation of simulation code from multimode models.

We show how the use of such single-mode methods can lead to the runtime errors observed on the Water Tank model shown above. We then introduce the exact multimode structural analysis performed by the IsamDAE tool, which will be used for the transformation of multimode models at the core of this article.

3.1 Approximate structural analysis

Structural analysis of a DAE system only relies on the knowledge of which numerical variables appear in which equations. As such, an approximate structural analysis of a multimode DAE system can be performed by abstracting away all mode dependencies inside the equations; for instance, an equation `x = if cond then y else z` will be regarded by the approximate structural analysis as an equation involving variables `x`, `y` and `z`.

Such an analysis of the Water Tank model shown in Figure 1 results in the decomposition shown in Figure 3. In this decomposition, equation `eh2` has to be solved for the variable `yh`.

When performing the pivoting of this equation, mode

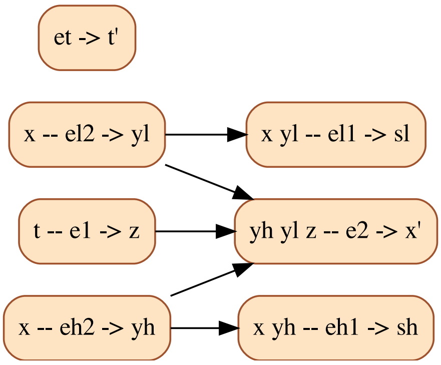


Figure 3. Dependency graph resulting from the approximate structural analysis of the Water Tank model. Vertices are equation blocks of the form $R - E \rightarrow W$, where: E is the block of equations; R is a set of variables to read (they are free variables, i.e., parameters of the block of equations); and W is a set of variables to write (they are the unknowns of the block of equations). When R is empty, the shorthand notation is $E \rightarrow W$. Edges express causal dependencies, meaning that a block can be solved only after all its predecessors have been solved.

dependencies have to be taken into account again. Equation eh2 reads:

$$0 = \texttt{if bh then x} - \texttt{xmax else yh}$$

which can be rewritten as an equation of the form $0 = \texttt{a yh} + \texttt{b}$ where a and b are mode-dependent:

$$\begin{aligned} 0 = & (\texttt{if bh then } 0 \texttt{ else } 1) \times \texttt{yh} \\ & + (\texttt{if bh then x} - \texttt{xmax else } 0) \end{aligned}$$

Unknown yh can finally be isolated:

$$\texttt{yh} = -\frac{\texttt{if bh then x} - \texttt{xmax else } 0}{\texttt{if bh then } 0 \texttt{ else } 1} \tag{1}$$

This technique may be used for the generation of simulation code, but in this case, a problem is bound to occur when Boolean variable bh is true. As a matter of fact, equation (1) is exactly the equation responsible for the division by zero exception shown in Figure 2, which occurs at the initial time, when bh is true.

3.2 Exact multimode structural analysis

The IsamDAE[1] tool (Caillaud, Malandain, and Thibault 2020) has been used to perform a multimode structural analysis of the model, resulting in the Conditional Dependency Graph (CDG) shown in Figure 4.

Remark that the differentiation index of the system is mode-dependent. For instance, equation el2 is used differentiated, to compute the derivative of x, when bl is true, while it is kept undifferentiated, to compute yl, when bl is false. Also notice that equation eh2 is no longer used to compute yh in all modes, but only when bh is false, thus preventing the runtime error explained above.

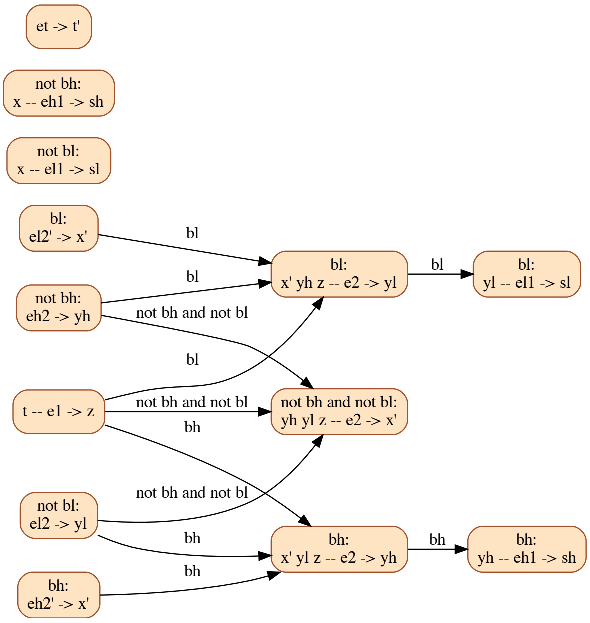


Figure 4. Conditional Dependency Graph resulting from the multimode structural analysis of the Water Tank model. Vertices are conditional equation blocks of the form $p : R - E \rightarrow W$, where: E is the block of equations; p is a Boolean condition, defining the set of modes in which the block has to be solved; R is a set of variables to read, or free variables, i.e., parameters of the block of equations; and W is a set of variables to write, meaning that they are the unknowns of the block of equations. When R is empty, the shorthand notation is $p : E \rightarrow W$. When p is the proposition true, it is omitted, and the notation becomes: $R - E \rightarrow W$, or $E \rightarrow W$. Edges express causal dependencies, meaning that a block can be solved only after all its predecessors have been solved. They are labeled by Boolean conditions, characterizing the modes in which the dependency applies.

We shall see next how the CDG (Figure 4) can be used to transform the model into an equivalent one, that triggers no runtime error when using Modelica tools based on an approximate structural analysis.

4 A Reduced Index Mode-Independent Structure (RIMIS) form

Using multimode structural analysis to transform a multimode Modelica model into a reduced-index model, that simulates correctly with state-of-the-art Modelica tools, is made difficult by the fact that the Modelica language does not permit to enable or disable an equation depending on the mode. Based on this limitation, the basic principle of our model transformation is to evaluate all equation blocks of the CDG in a mode-independent fashion, irrespectively of the mode in which the system is. Of course, this leads to useless computations during simulation. However, this

[1] https://team.inria.fr/hycomes/software/isamdae/

turns out to be a systematic way to ensure a correct simulation of multimode Modelica models.

The method proposed in this paper is detailed below, in informal terms, then illustrated on a simple example. A mathematical definition of the transformation is detailed in Section 6. Remark that models with initial equations, `when` or `reinit` statements are not covered in this paper. Also note that models with non-scalar variables or class instances of any kind are not considered here. It is assumed that the models have been flattened according to the procedure described in Chapter 5 of the Modelica Language Specification (The Modelica Association 2021). Because of a current restriction of the IsamDAE software, mode variables are assumed to be of type Boolean.

4.1 The RIMIS form transformation

The method decomposes in the following seven steps:

1. **Conditional Dependency Graph:** The CDG of the source model is computed by the multimode structural analysis method. This graph defines a block-triangular decomposition of the reduced-index system, for each mode of the system. It will be used throughout the transformation.

2. **Source Variables:** Variable declarations are copied unchanged, with the exception of real variables, whose initialization parts are removed.

3. **Replicate and Dummy Derivative Variables:** For each block of the CDG, replicates of written variables (unknowns) are declared. Whenever an unknown appears differentiated, a dummy derivative variable (Mattsson and Soderlind 1993) is declared. Initialization statements for state variables are copied from the source model. As an optional optimization, non-leading replicate variables can be factored among a disjunction of modes, in order to decrease the number of variables in the resulting model.

4. **Mode Equations:** Equations defining mode variables are copied unchanged. For the sake of simplicity, these equations are assumed to be of the form `b` $= (expr >= 0)$, where *expr* is a real expression.

5. **Replicate and Dummy Equations:** Equations are replaced with replicates, according to the following principle:

   For each block in the CDG, equations appearing in this block are replicated, substituting (i) every written variable (unknown of the block) by the replicate declared in step 3, and (ii) every read variable (parameter of the block) by the corresponding replicate, if it is a leading variable. Both mode variables and read state variables are left unchanged.

   As a result, the single-mode structural analysis of the resulting equation system yields a block-triangular decomposition that contains all the blocks of the CDG obtained by the multimode structural analysis of the original model.

   For each equation in the fresh model, the propositional formula conditioning the block in which this equation appears can be taken into account: a partial evaluation of the equation is performed (Jones, Gomard, and Sestoft 1993). This has the effect of simplifying the equation, by eliminating some of the conditionals (`if` ... `then` ... `else` ... operators).

   Note that the resulting equations may still be multimode: in general, not all conditionals can be eliminated by partial evaluation. However, the fact that the structure of the resulting equations is independent of the mode is still guaranteed: the multimode structural analysis ensures that each equation block has the same structure (in particular, the same read and written variables) in all the modes in which it is defined, even if one or several of its equations contain conditional statements.

   First-order differential equations are also added in accordance to the dummy derivatives method.

6. **Multiplexing Equations:** In order to retrieve the values of the source model variables from the replicates in the fresh model, mutiplexing equations have to be added. These are multimode equations, containing conditional operators, but these equations contain no dynamics: each multiplexing equation focuses on a source model variable that corresponds to several replicates in the transformed model, specifying which of the latter currently holds the value of the former.

7. **Reinitializations:** Reinitialization statements finally have to be inserted, in order to reset replicate variables that are state variables to a correct value upon the occurrence of a mode switching. Therefore, these statements are triggered by mode changes.

4.2 Transformation of a simple model

We illustrate the method on a simplistic, yet relevant, two equations model:

```
model TwoEquations
  Real x(start=0,fixed=true);
  Boolean p(start=false,fixed=true);
equation
  p = (x >= 1);
  1 = if p then x else der(x);
end TwoEquations;
```

This model has one real equation, one Boolean equation, and no particular physical meaning. However, it captures in a nutshell the difficulty raised with the Water Tank system. As a matter of fact, the CDG (Figure 5) resulting from the multimode structural analysis distinguishes between two cases:

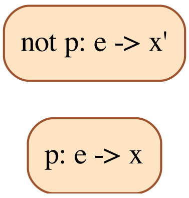
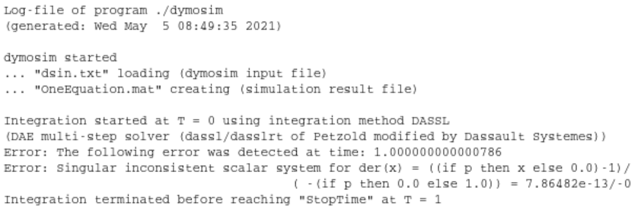


Figure 5. CDG of the Two Equations model.



```
Log-file of program ./dymosim
(generated: Wed May  5 08:49:35 2021)

dymosim started
... "dsin.txt" loading (dymosim input file)
... "OneEquation.mat" creating (simulation result file)

Integration started at T = 0 using integration method DASSL
(DAE multi-step solver (dassl/dasslrt of Petzold modified by Dassault Systemes))
Error: The following error was detected at time: 1.000000000000786
Error: Singular inconsistent scalar system for der(x) = ((if p then x else 0.0)-1)/
                                        ( -(if p then 0.0 else 1.0)) = 7.86482e-13/-0
Integration terminated before reaching "StopTime" at T = 1
```

Figure 6. Failed simulation of the Two Equations model with Dymola 2021.

- when `p` is `true`, `x` is a leading variable, meaning that it is the unknown that needs to be solved;
- when `p` is `false`, the leading variable is x', the first-order time derivative of `x`, while `x` itself is a state variable.

The approximate structural analysis of both Dymola and OpenModelica determines that the leading variable is x' in all modes; however, the real equation is singular in x' when `p` is `true`. Unsurprisingly, an exception is raised during simulation, as shown in Figure 6.

Let us apply the transformation one step after the other:

1. The **CDG graph** of the source model is shown in Figure 5.
2. **Declarations** of variables `x` and `p` are copied.

   ```
   Real x;
   Boolean p(start=false,fixed=true);
   ```

   Remark that the declaration of `x` has been stripped of its initialization part.

3. **Replicate variables** are created according to the two blocks of the CDG. Two leading replicate variables `x_2` (holding the value of `x` if `p` holds) and `x_p_3` (holding the value of x' if `not p` holds), and one state replicate variable `x_3` that is meaningful only if `not p` holds, are declared.

   ```
   Real x_2;
   Real x_p_3;
   Real x_3(start=0,fixed=true);
   ```

   Note that the initialization of variable `x` in the source model is copied here, to initialize the replicate state variable `x_3`.

4. One **mode equation** is copied from the source model.

   ```
   p = (x >= 1);
   ```

5. **Replicate equations** are generated from the CDG, which has two blocks of one equation each.

   From the block $\mathtt{p} : \mathtt{e} \to \mathtt{x}$, one replicate equation is generated by replacing variable `x` with its replicate `x_2`, then performing the partial evaluation (Jones, Gomard, and Sestoft 1993) under the assumption that the Boolean condition `p` holds.

   ```
   // Block e_2 -> x_2
   /* e_2 : */ 1 = x_2;
   ```

   From the second block $\mathtt{not\ \ p} : \mathtt{e} \to \mathtt{x}'$, one replicate equation is generated in a similar way.

   ```
   // Block e_3 -> x_p_3
   /* e_3 : */ 1 = x_p_3;
   ```

   A differential equation is also generated, linking replicate variable `x_3` with its dummy derivative `x_p_3`.

   ```
   der(x_3) = x_p_3;
   ```

6. One **multiplexing equation** is generated, to be solved for variable `x`.

   ```
   x = if p then x_2 else x_3;
   ```

7. Finally, the only case in which a state variable has to be **reinitialized** is when entering the mode `not p`. The value of replicate variable `x_3` is then set to be the left limit of `x`.

   ```
   when not p then
     reinit(x_3,pre(x));
   end when;
   ```

The complete RIMIS form of the Two Equations model is given in Figure 7. The result of the successful simulation of this model is shown in Figure 8. Remark that the mode switching from $\mathtt{p} = \mathtt{false}$ to $\mathtt{p} = \mathtt{true}$ is correct, and that the reinitialization statement is never evaluated, as `p` remains `true` forever after time $\mathtt{t} = 1$.

5 Successful simulations of the Water Tank system in RIMIS form

The RIMIS transformation is illustrated on the Water Tank model (Figure 1); the resulting model is shown in Figure 9. Simulation results obtained with Dymola 2021 are shown in Figure 10. It can be seen that the simulation is successful, with a correct behavior of the Water Tank system, while the simulation of the original model failed (Figure 2). A correct simulation has also been obtained with OpenModelica 1.17.0 (Fritzson et al. 2020), under the provision that the Newton solver is used instead of the KINSOL nonlinear solver.

```
model TwoEquations_rimis
  // Source variables
  Real x;
  Boolean p(start=false,fixed=true);
  // Replicate variables
  Real x_2;
  Real x_p_3;
  Real x_3(start=0,fixed=true);
equation
  // Mode equations
  p = (x >= 1);
  // Differential equations
  der(x_3) = x_p_3;
  // Multiplexing
  x = if p then x_2 else x_3;
  // Block e_3 -> x_p_3
  /* e_3 : */ 1 = x_p_3;
  // Block e_2 -> x_2
  /* e_2 : */ 1 = x_2;
  // Replicate reinitializations
  when not p then
    reinit(x_3,pre(x));
  end when;
end TwoEquations_rimis;
```

Figure 7. Two Equations model in RIMIS form.

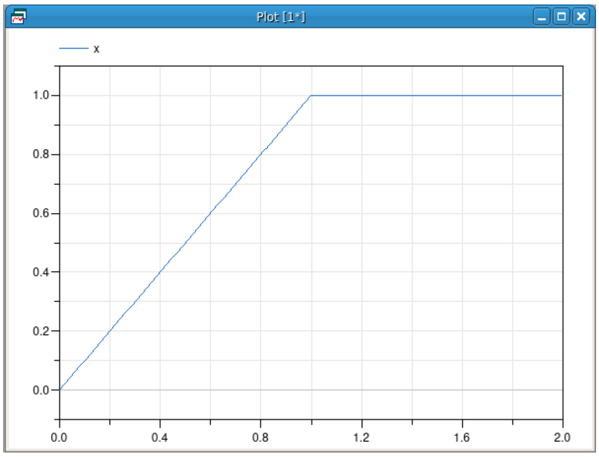


Figure 8. Simulation of the Two Equations model in RIMIS form with Dymola 2021.

6 Formalizing the RIMIS form transformation

The mathematical definition of the RIMIS form transformation relies on the partial evaluation of equations. Once variable renaming is also properly defined, the seven-step transformation mentioned in Section 4.1 is formalized. Finally, an optimization aiming at reducing the transformed model is presented.

6.1 Partial evaluation of expressions and equations

Partial evaluation is an umbrella name for a set of program transformation techniques that aim at specializing a program by taking into account prior knowledge on its input data, possibly improving its performances (Jones, Gomard, and Sestoft 1993; Danvy, Glück, and Thiemann 1996).

In the context of the Modelica language, consider a Boolean expression q, and a real expression e. The partial evaluation of expression e, assuming q, is an expression $e' = \pi_q(e)$, such that q implies $e = e'$ and $\mathrm{free}(e') \subseteq \mathrm{free}(e)$, where $\mathrm{free}(.)$ is the set of free variables appearing in an expression.

To define the partial evaluation operator π, and for the sake of clarity, we only consider the subset of the Modelica expression language defined by the following grammar, where p is a Modelica Boolean expression:

$$\begin{array}{lcll} e & ::= & c & \text{where } c \text{ is a constant} \\ & | & e \text{ op } e & \text{where op} \in \{+,-,*,\ldots\} \\ & | & v & \text{where } v \text{ is an identifier} \\ & | & v(e,\ldots e) & \\ & | & \texttt{if } p \texttt{ then } e \texttt{ else } e & \end{array}$$

Given a Boolean expression q and a real expression e, the partial evaluation of e, assuming q, is defined by induction on the structure of e:

$$\begin{cases} \pi_q(c) & \equiv \quad c \\ \pi_q(e_1 \text{ op } e_2) & \equiv \quad \pi_q(e_1) \text{ op } \pi_q(e_2) \\ \pi_q(v) & \equiv \quad v \\ \pi_q(v(e_1,\ldots e_n)) & \equiv \quad v(\pi_q(e_1),\ldots \pi_q(e_n)) \\ \pi_q(\texttt{if } p \texttt{ then } e_T \texttt{ else } e_F) & \equiv \quad \mathrm{cond}_q(p,e_T,e_F) \end{cases}$$

where

$$\begin{array}{l|ll} \multicolumn{3}{l}{\mathrm{cond}_q(p,e_T,e_F) \equiv} \\ & \pi_{q \texttt{ and } p}(e_T) & \text{if } q \texttt{ and not } p \\ & & \text{is unsatisfiable, else} \\ & \pi_{q \texttt{ and not } p}(e_F) & \text{if } q \texttt{ and } p \\ & & \text{is unsatisfiable, else} \\ & \texttt{if } r & \text{where } r \text{ is such that:} \\ & \texttt{ then } \pi_{q \texttt{ and } p}(e_T) & p \texttt{ and } q \text{ implies } r \text{, and} \\ & \texttt{ else } \pi_{q \texttt{ and not } p}(e_F) & r \text{ implies } p \texttt{ or not } q \end{array}$$

In the above definition, condition r is not unique: whenever possible, it should be chosen such that it is more concise than p.

The extension of the partial evaluation operator to equations is straightforward:

$$\pi_q(e_{LHS} = e_{RHS}) \quad \equiv \quad \pi_q(e_{LHS}) = \pi_q(e_{RHS}) \quad .$$

6.2 Variable renaming

Before moving to the formal definition of the RIMIS transformation, variable renaming must be defined, in order to declare replicate variables and transform equations into their replicates.

Given a Boolean expression p, an identifier v, and a differentiation order $n \geq 0$, the replicate of the n-th order derivative of v, under condition p, is the identifier $\rho^n_p(v)$.

```
model WaterTankRIMIS
  // Constants
  constant Real xmax = 1.0;
  constant Real xmin = 0.0;
  constant Real y0 = 6.667;
  constant Real rho = 0.8;
  // Variables
  Real t(start=0,fixed=true);
  Real x(start=0.5,fixed=true);
  Real yh;
  Real yl;
  Real z;
  Real sh;
  Real sl;
  Boolean bh(start=false,fixed=true);
  Boolean bl(start=false,fixed=true);
  // Dummy derivatives
  Real t_p;
  Real x_p;
  // Replicated algebraic variables
  Real sh_5; // sh if not bh
  Real sh_6; // sh if bh
  Real sl_2; // sl if not bl
  Real sl_4; // sl if bl
  Real x_p_4; // x' if bl
  Real x_p_7; // x' if not bh and not bl
  Real x_p_6; // x' if bh
  Real yh_5; // yh if not bh
  Real yh_6; // yh if bh
  Real yl_2; // yl if not bl
  Real yl_4; // yl if bl
equation
  // Boolean equations
  bh = (sh >= 0);
  bl = (sl >= 0);
  // Differential equations
  der(t) = t_p;
  der(x) = x_p;
  // Multiplexing equations
  yh = if bh then yh_6 else yh_5;
  yl = if bl then yl_4 else yl_2;
  sh = if bh then sh_6 else sh_5;
  sl = if bl then sl_4 else sl_2;
  x_p = if bh then x_p_6 else
          if bl then x_p_4 else x_p_7;
  // Block et -> t'
  t_p = 1;
  // Block not bh: x -- eh1 -> sh
  sh_5 = x - xmax;
  // Block not bl: x -- el1 -> sl
  sl_2 = xmin - x;
  // Block bl: el2' -> x'
  x_p_4 = 0;
  // Block not bh: eh2 -> yh
  yh_5 = 0;
  // Block x -- e1 -> z
  z = rho*y0*(1+
      Modelica.Math.cos(2*Modelica.Constants.pi*t));
  // Block not bl: el2 -> yl
  yl_2 = 0;
  // Block bh: eh2' -> x'
  x_p_6 = 0;
  // Block bl: x' yh z -- e2 -> yl
  yl_4 = y0 + x_p_4 + yh_5 - z;
  // Block not bh & not bl: yh yl z -- e2 -> x'
  x_p_7 = z + yl_2 - yh_5 - y0;
  // Block bh: x' yl z  -- e2 -> yh
  yh_6 = z + yl_2 - x_p_6 - y0;
  // Block bl: yl -- el1 -> sl
  sl_4 = yl_4;
  // Block bh: yh -- eh1 -> sh
  sh_6 = yh_6;
end WaterTankRIMIS;
```

Figure 9. The Water Tank system in RIMIS form.

The operator ρ is assumed to satisfy the following axioms:

$$
\begin{array}{lll}
(\textbf{Identity}) & \rho^{0}_{\texttt{true}}(u) = u & \\
(\textbf{Injectivity}) & \rho^{n}_{p}(u) = \rho^{m}_{q}(v) \text{ implies} & u = v \text{ and} \\
 & & p \iff q \text{ and} \\
 & & n = m
\end{array}
$$

Checking the equivalence of two Boolean expressions is, in general, a difficult problem. In this article, Boolean expressions that appear in conditional statements are restricted to propositional formulas only. Mode equations are restricted to the form $v = (e >= 0)$, where e is an affine expression. Under these assumptions, equivalence checking can be done with BDDAPRON, a logico-numerical abstract domain library (Jeannet 2012) combining BDDs (Boolean Decision Diagrams) (Bryant 1986) and polyhedra (Schrijver 1998). Such a use of BDDAPRON is considered, among other program analyses, in Chapter 7 of (Schrammel 2012).

6.3 Formal definition of the RIMIS form transformation

Consider a Modelica model M that can be decomposed in the following parts:

$$M \;\equiv\; \mathrm{MD} \uplus \mathrm{RD} \uplus \mathrm{RI} \uplus \mathrm{ME} \uplus \mathrm{RE}$$

where:

- MD is the set of **m**ode (Boolean) variable **d**eclarations and initializations;
- RD is the set of **r**eal variable **d**eclarations, *stripped of their initializations*;
- RI is the set of **r**eal variable **i**nitializations;
- ME is the set of **m**ode variable **e**quations;
- RE is the set of **r**eal **e**quations.

Remark that models with `when` and `reinit` statements are not covered by the RIMIS form transformation, as this

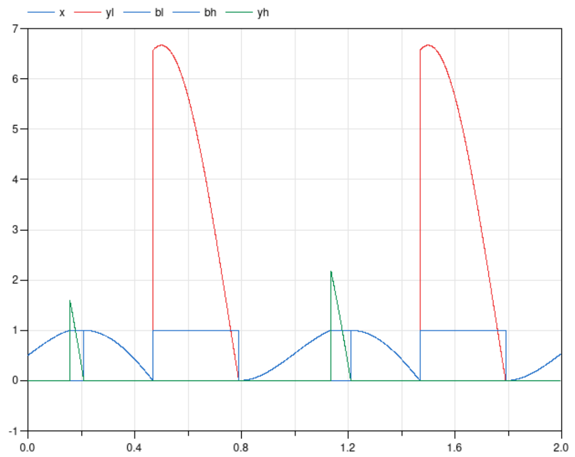


Figure 10. Simulation of the Water Tank system in RIMIS form with Dymola 2021.

would require a multimode structural analysis of mode changes (Benveniste, Caillaud, and Malandain 2020), that is not yet implemented in the IsamDAE software (Caillaud, Malandain, and Thibault 2020). Because of a current restriction of IsamDAE, mode variables are assumed to be Boolean.

Model M is assumed to be structurally nonsingular in all modes. Its CDG computed by the multimode structural analysis (Caillaud, Malandain, and Thibault 2020) consists in a set of blocks of equations and a set of directed edges between blocks; let Blocks and Edges denote the corresponding sets. A block $b \in \text{Blocks}$ consists of four parts:

- $\text{cond}(b)$, a Boolean expression;
- $\text{Eqs}(b)$, a set of equations, possibly differentiated;
- $\text{Read}(b)$, a set of read variables (parameters of the block of equations);
- $\text{Write}(b)$, a set of written variables (unknowns of the block of equations).

Elements of $\text{Eqs}(b)$ are pairs of the form $(0 = e, k)$, where e is an expression and $k \geq 0$ is a differentiation order. Elements of $\text{Read}(b)$ and $\text{Write}(b)$ are pairs of the form (u, k), where u is an identifier and $k \geq 0$ is a differentiation order. An edge $g \in \text{Edges}$ consists of three parts:

- $\text{cond}(g)$, a Boolean expression;
- $\text{from}(g),\ \text{to}(g) \in \text{Blocks}$, two blocks.

The meaning of an edge g is that whenever $\text{cond}(g)$ holds, block $\text{from}(g)$ has to be solved before block $\text{to}(g)$. By construction, $\text{cond}(g)$ implies both $\text{cond}(\text{from}(g))$ and $\text{cond}(\text{to}(g))$.

In addition, the multimode structural analysis computes several functions and predicates on (differentiated) variables $v = (u, k)$:

- $\text{leading}_p(v)$ decides whether variable u is a leading variable in some mode satisfying the Boolean formula p;
- $\text{algebraic}_p(v)$ decides whether u is an algebraic variable in some mode satisfying p;
- $\text{state}_p(v)$ decides whether u is a state variable in some mode satisfying p.

For the sake of clarity, the following notations are introduced: $\text{leading}(b) = \{v \in \text{Read}(b) \cup \text{Write}(b) | \text{leading}_{\text{cond}(b)}(v)\}$ is the set of leading variables appearing in block b; $\text{Def}_p(v)$ is the set of blocks that define variable v in some mode satisfying the Boolean formula p, either because v itself is written, or because a higher order derivative of it is written:

$$\begin{aligned}\text{Def}_p(u,k) \quad = \quad & \{b \in \text{Blocks} \mid p \wedge \text{cond}(b) \text{ is satisfiable,} \\ & \text{and } \exists k' \geq k,\ (u,k') \in \text{Write}(b)\}\end{aligned}$$

The resulting RIMIS form model can be decomposed in several parts:

$$\begin{aligned}\text{RIMIS} \quad \equiv \quad & \text{MD} \uplus \text{RD} \uplus \text{DECL} \uplus \text{INIT} \uplus \\ & \text{ME} \uplus \text{REPL} \uplus \text{MULTI} \uplus \text{DIFF} \uplus \text{REINIT}\end{aligned}$$

where:

- MD is the set of **m**ode (Boolean) variable **d**eclarations and initializations, taken from M;
- RD is the set of **r**eal variable **d**eclarations, taken from M;
- DECL is the set of replicate variable **decl**arations, defined below;
- INIT is the set of replicate variable **init**ializations, defined below;
- ME is the set of **m**ode variable **e**quations, taken from M;
- REPL is the set of **repl**icate equations, defined below;
- MULTI is the set of **multi**plexing equations, defined below;
- DIFF is the set of **diff**erential equations, defined below;
- REINIT is the set of **reinit**ialization equations, defined below.

Replicate variable declarations (Section 4.1, step 3) consist in the declaration of the following set of real variables:

$$\begin{aligned}\text{DECL} \quad \equiv \quad & \bigcup_{b \in \text{Blocks}, (u,k) \in \text{Read}(b) \cup \text{Write}(b)} \\ & \left\{\rho^i_{\text{cond}(b)}(u) \mid 0 \leq i \leq k\right\} \quad .\end{aligned}$$

Replicate variable initializations (Section 4.1, step 3) consist in the initialization of all replicate variables $\rho^0_{\text{cond}(b)}(u)$ that are state variables, with the initialization expression for u in M ($\text{RI}(u)$):

$$\text{INIT} \equiv \left\{(\rho^0_p(u), \text{RI}(u)) | \rho^0_p(u) \in \text{DECL and state}_p(u,0)\right\}$$

where ρ is a fixed replication operator as defined in Section 6.2.

Replicate equations (Section 4.1, step 5) consist in the differentiation to a given order of the equations of each block of equations:

$$\begin{array}{lcl} \text{REPL} & \equiv & \bigcup_{b\in\text{Blocks}} \\ & & \left\{\sigma_b(\pi_{\text{cond}(b)}(\delta_k(q))) \mid (q,k) \in \text{Eqs}(b)\right\} \end{array}$$

where π is the partial evaluation operator defined in Section 6.1, equation $\delta_k(q)$ is the k-th order differentiation of equation q, and σ_b is the substitution operator such that $\sigma_b(q)$ substitutes any variable u in equation q with the replicate variable $\rho^0_{\text{cond}(b)}(u)$, any derivative of the form $\texttt{der}(u)$ by the replicate variable $\rho^1_{\text{cond}(b)}(u)$, and so on for higher order derivatives.

Multiplexing equations (Section 4.1, step 6) serve two purposes: (i) linking written variables and read variables in different blocks, and (ii) defining the original real variables from M:

$$\begin{array}{lcl} \text{MULTI} & = & \bigcup_{b\in\text{Blocks}, v=(u,k)\in\text{Read}(b)} \\ & & \{\rho^k_{\text{cond}(b)}(u) = \text{case}_v(\text{Def}_{\text{cond}(b)}(v))\} \cup \\ & & \bigcup_{u\in\text{RD}}\{u = \text{case}_{u,0}(\text{Def}_{\texttt{true}}(u,0)\} \end{array}$$

where case_v is defined by induction over the set of blocks $\text{Def}_{\texttt{true}}(v)$ that define variable v in some mode:

$$\begin{array}{lcl} \text{case}_{(u,k)}(\{b\}) & = & \rho^k_{\text{cond}(b)}(u) \\ \text{case}_{v=(u,k)}(b \uplus B) & = & \texttt{if } \text{cond}(b) \\ & & \quad \texttt{then } \rho^k_{\text{cond}(b)}(u) \\ & & \quad \texttt{else } \text{case}_v(B) \end{array}$$

Differential equations (Section 4.1, step 5) serve the purpose of defining replicate state variables from the replicate dummy derivatives:

$$\begin{array}{lcl} \text{DIFF} & = & \bigcup_{b\in\text{Blocks},(u,k)\in\text{Write}(b)} \\ & & \{\,\texttt{der}(\rho^i_{\text{cond}(b)}(u)) = \rho^{i+1}_{\text{cond}(b)}(u)\,\}_{0\le i\le k-1} \end{array}$$

Finally, upon the occurrence of a mode change, **reinitialization statements** (Section 4.1, step 7) serve the purpose of copying the state vector from a formerly active replicate state variable to a newly active one:

$$\begin{array}{lcl} \text{REINIT} & = & \bigcup_{b\in\text{Blocks},(u,1)\in\text{Write}(b)} \\ & & \{\texttt{when } \text{cond}(b) \texttt{ then} \\ & & \quad \texttt{reinit(}\ \rho^0_{\text{cond}(b)}(u)\ \texttt{, pre}(u)\texttt{);} \\ & & \texttt{endwhen}\} \end{array}$$

6.4 Optimization

Modelica code generated with the procedure described in Section 6.3 may contain multiplexing equations and reinitialization statements that can be eliminated thanks to the optimization described below.

It may happen that a multiplexing equation is of the form $\rho^k_p(u) = \rho^k_{p'}(u)$. This typically happens when a block $b \in$ Blocks reads a variable that is written by exactly one block $b' \in$ Blocks. In this case, no multiplexing equation needs to be generated, and replicate variable $\rho^k_p(u)$ does not need to be declared. Instead, every occurrence of $\rho^k_p(u)$ in equations $q \in \text{Eqs}(b)$ shall be replaced by $\rho^k_{p'}(u)$.

Remark that this optimization has been applied to the Water Tank model in RIMIS form (Figure 9). For instance, equation `sh_5` = `x` − `xmax` refers directly to variable `x` instead of variable `x_5`, sparing both the declaration of the replicate variable `x_5` and the generation of the multiplexing equation `x` = `x_5`. The same optimization has been applied to variable `z`.

7 Conclusion

We presented a method for transforming multimode Modelica models that yield simulation errors with state-of-the-art Modelica tools (such as Dymola 2021 and OpenModelica 1.17.0) into Reduced Index Mode-Independent Structure (RIMIS) models that simulate correctly with the same tools.

This model transformation relies on the multimode structural analysis as performed by the IsamDAE tool (Caillaud, Malandain, and Thibault 2020). The output of this structural analysis, which is a Conditional Dependency Graph (CDG) describing all possible equation blocks in all modes and their dependencies, is used to replicate equations and real variables as needed. This is performed in such a way that the approximate structural analysis implemented in most Modelica tools will create the same equation blocks. Dummy derivatives (Mattsson and Soderlind 1993) are also used so that the resulting model is of index 0.

The 7-step RIMIS transformation was detailed on a very simple multimode model, then applied to the Modelica model of a water tank system; we showed that, while both source models cause division by zero errors at runtime, their RIMIS forms simulate correctly with both Dymola 2021 and OpenModelica 1.17.0, yielding the expected behaviors for their variables. This process was formalized, paving the way towards its automation for the handling of a wider class of multimode models by state-of-the-art Modelica tools.

A possible drawback of this approach is that the size of the RIMIS model may *a priori* be exponential in the size of the source model, as both equations and real variables could be replicated once for every mode of the system. However, experiments on a number of parametric models with the IsamDAE tool show that the number of blocks in the CDG of such models tend to be linear in their size, except for rare pathological cases. As such, the size of the RIMIS form of a multimode Modelica model will, in a vast majority of cases, be linear in the size of the original model, thus making our approach tractable even for large models.

As a concluding remark, it can be noted that the illustrative models in this article are only made of linear equations, so that the evaluation of all equation blocks, both active and inactive, at every time step is not an issue. For nonlinear blocks, not only could this approach be computationally expensive, but it might fail altogether, as such blocks might be singular outside of a given subset of the modes.

A simple fix, that was not detailed above, consists in transforming the equations from such blocks into conditional equations, so that they become trivial equations outside of the set of modes in which they have to be considered. The matching between equations and variables that is computed during the multimode structural analysis can be used for this task, as it basically tells 'which variable has to be solved using which equation'; a nonlinear equation could then be replaced with the simple assignment of a default value to its matched real variable in the modes in which the equation block is inactive. This additional transformation would still preserve the structure of the model, in the sense that the approximate structural analysis would still result in solving the same blocks for the same real variables.

Acknowledgements

This work was supported by the FUI ModeliScale DOS0066450/00 French national collaborative project, the Glose Inria-Safran Tech bilateral collaboration, and the Inria IPL ModeliScale large scale initiative (`https://team.inria.fr/modeliscale/`).

References

Benveniste, Albert, Benoit Caillaud, Hilding Elmqvist, et al. (2019). "Multi-Mode DAE Models - Challenges, Theory and Implementation". In: *Computing and Software Science - State of the Art and Perspectives*. Ed. by Bernhard Steffen and Gerhard J. Woeginger. Vol. 10000. Lecture Notes in Computer Science. Springer, pp. 283–310. ISBN: 978-3-319-91907-2. DOI: 10.1007/978-3-319-91908-9_16.

Benveniste, Albert, Benoit Caillaud, and Mathias Malandain (2020). "The mathematical foundations of physical systems modeling languages". In: *Annual Reviews in Control* 50, pp. 72–118. ISSN: 1367-5788. DOI: 10.1016/j.arcontrol.2020.08.001.

Benveniste, Albert, Benoit Caillaud, and Mathias Malandain (2021-09). "Handling Multimode Models and Mode Changes in Modelica". In: *Proceedings of the 14th International Modelica Conference*. Linköping University Electronic Press.

Bryant, Randal E. (1986). "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* 35, pp. 677–691.

Caillaud, Benoit, Mathias Malandain, and Joan Thibault (2020-04). "Implicit Structural Analysis of Multimode DAE Systems". In: *23rd ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2020)*. Sydney, Australia. DOI: 10.1145/3365365.3382201.

Campbell, Stephen L. and C. William Gear (1995). "The index of general nonlinear DAEs". In: *Numer. Math.* 72, pp. 173–196.

Danvy, Olivier, Robert Glück, and Peter Thiemann, eds. (1996). *Partial Evaluation, International Seminar, Dagstuhl Castle, Germany, February 12-16, 1996, Selected Papers*. Vol. 1110. Lecture Notes in Computer Science. Springer. ISBN: 3-540-61580-6. DOI: 10.1007/3-540-61580-6.

Dassault Systèmes AB (2020). *Dymola official webpage*. Accessed: 2021-06-28. URL: https://www.3ds.com/products-services/catia/products/dymola/.

Elmqvist, Hilding et al. (2012-09). "State Machines in Modelica". In: *Proc. of the Int. Modelica Conference*. Ed. by Martin Otter and Dirk Zimmer. Modelica Association. Munich, Germany, pp. 37–46.

Fritzson, Peter et al. (2020). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Giorgidze, George and Henrik Nilsson (2011). "Embedding a Functional Hybrid Modelling Language in Haskell". In: *Implementation and Application of Functional Languages*. Ed. by Sven-Bodo Scholz and Olaf Chitil. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 138–155. ISBN: 978-3-642-24452-0.

Jeannet, Bertrand (2012-08). *BddApron*. URL: http://pop-art.inrialpes.fr/~bjeannet/bjeannet-forge/bddapron/.

Jones, Neil D., Carsten K. Gomard, and Peter Sestoft (1993). *Partial evaluation and automatic program generation*. Prentice Hall international series in computer science. Prentice Hall. ISBN: 978-0-13-020249-9.

Mattsson, Sven Erik and Gustaf Soderlind (1993). "Index Reduction in Differential-Algebraic Equations Using Dummy Derivatives". In: *SIAM Journal on Scientific Computing* 14.3, pp. 677–692. DOI: 10.1137/0914043.

Nilsson, Henrik and George Giorgidze (2010). "Exploiting structural dynamism in Functional Hybrid Modelling for simulation of ideal diodes". In: *Czech Technical University Publishing House*.

Pantelides, Constantinos C. (1988). "The consistent initialization of differential-algebraic systems". In: *SIAM J. Sci. Stat. Comput.* 9.2, pp. 213–231.

Pryce, John D. (2001). "A simple structural analysis method for DAEs". In: *BIT* 41.2, pp. 364–394.

Schrammel, Peter (2012). "Méthodes logico-numériques pour la vérification des systèmes discrets et hybrides. (Logico-Numerical Verification Methods for Discrete and Hybrid Systems)". PhD thesis. Grenoble Alpes University, France. URL: https://tel.archives-ouvertes.fr/tel-00809357.

Schrijver, A. (1998-04). *Theory of linear and integer programming*. Wiley.

The Modelica Association (2021-02). *Modelica, A Unified Object-Oriented Language for Systems Modeling. Language Specification, Version 3.5*. URL: https://www.modelica.org.

Van Der Schaft, A. J. and J. M. Schumacher (1998). "Complementarity modeling of hybrid systems". In: *IEEE Transactions on Automatic Control* 43.4, pp. 483–490. DOI: 10.1109/9.664151.

Zimmer, Dirk (2010). "Equation-Based Modeling of Variable-Structure Systems". PhD thesis. ETH Zürich, No. 18924.

Evaluating a Tree Diff Algorithm for Use in Modelica Tools

Martin Sjölund[1]

[1]Department of Computer Science, Linköping University, Sweden, martin.sjolund@liu.se

Abstract

Modelica tools change the formatting of the source code when performing operations in the graphical user interface. These unintended changes cause problems for source code management since a code review would mostly go through changes that do not change any semantics. The intent of this work is to present a workflow where edits from an interactive graphical user interface does not contain these unintended changes when using the source code management system.

A diff tool that can merge two Modelica files and produce a merged copy is presented and evaluated. The diff algorithm works by comparing syntax subtrees of Modelica code and having some domain knowledge about which subtrees belong together, speeding up the diff algorithm. The result is a merged file by taking formatting of the first file and the semantics from the second file. This works very well for smaller changes (a single edit) and scales with file size (making the user interface faster for smaller files).

To test the algorithm on a larger set of changes, a conversion script was applied to a set of libraries. The effect of applying a conversion script is a set of automated edit operations, which cause unintended changes in the formatting of the source code. The diff algorithm with applied to these changes and the performance was analyzed.

The results are very promising especially for Modelica libraries that are split into multiple files rather than a large single file. Having a single large file takes slightly longer to process and produces additional unintended formatting changes compared to a library developed as a set of smaller files.

Keywords: Modelica, diff, file comparison, conversion script, interactive user interface

1 Introduction

An important problem to handle for any software development is the management of source code. It is important to be able to see what changes are introduced in every new version of your software and one common way of showing this is with a simple text diff. However, most text diff tools are very limited in what they can do and introducing any new whitespace in a line will often flag the entire line. If you introduce a line break or move part of a line around, you can often forget about seeing what you actually changed in a diff. Most text diff algorithms will use something similar to Myers (1986) algorithm, performing the diff on lines of code since its $O(ND)$ scaling performs poorly on when words are used items instead of lines. There are algorithms available to improve the performance of Myers (1986), such as diff-match-patch (Google 2019), but they have most of the same drawbacks as the original.

When working with source code management systems such as git, there are best practices to make the history easier to read. Some of the following best practices make it easier for Myers (1986) algorithm to work since there are fewer changes to consider:

- every developer must use the same width for indentation (tabular characters and/or spaces).
- include no trailing whitespace at the ends of line.
- use text line endings (to avoid CR/LF issues).
- don't commit generated files such as binaries.
- commit only related work together.

Tools that compare source code of languages do exist, for example Diff/TS (Hashimoto and Mori 2008) and GumTree, but these need domain knowledge about the language to perform a diff on and need to be tuned to produce good results (Matsumoto, Higo, and Kusumoto 2019).

How this problem relates to the Modelica language (Modelica Association 2021) is probably apparent to anyone who has collaborated on a Modelica project. Modelica tools are graphical user interfaces where you move components around, change some value, or drag and drop components. This means that the source code needs to be added, removed, or updated. If these operations are performed internally to the Modelica tools, the internal representation needs to be unparsed in order to write these changes to file[1]. This unparsing will be slightly different in all Modelica tools, and some tools may be smart enough to at least only update the part of the class that changed. In this work, OpenModelica (Fritzson et al. 2020) and its graphical user interface OMEdit will be used to evaluate the work. OpenModelica will update the entire class and it will move something around since the internal representation for example only allows comments in certain places (compare Listings 1 and 2).

[1] The edits could also be performed directly on a concrete syntax tree, but this would require a full redesign of how Modelica code is handled in Modelica tools and would be much harder to implement.

DOI 10.3384/ecp21181529

```
model M
  MyModel m(
    // ??? Works if we do this
    x = 0,
    // Disable heat port
    y = false,
    // Forces the model into mode 2
    z = 200
  );
end M;
```

Listing 1. Example listing with comment.

```
model M
  MyModel m(x = 0, y = false, z = 200);
  // ??? Works if we do this
  // Disable heat port
  // Forces the model into mode 2
end M;
```

Listing 2. The example in Listing 1 unparsed by OpenModelica moves the comment.

What this paper tries to answer is how to perform a diff of unparsed Modelica code in a way that would produce small text diffs in a source code management system.

In order to be able to evaluate the proposed solution, a large enough test set is required. The Modelica Language Specification 3.4 (Modelica Association 2017) standardized the concept of conversion scripts. Using conversion scripts it is possible to for example rename a component in a class of a library and to automatically upgrade a model using the old version of the library with the new names. This potentially changes every single line of code in a library that is converted in this way. The Modelica Standard Library version 4.0.0[2] has a conversion script from major version 3 and there are many libraries still using one of these versions of the standard library. The diff algorithm can be evaluated by applying the conversion script to these libraries since a large set of real-world edit operations will be produced.

2 Method

The Modelica diff algorithm was created over several iterations. The current implementation will be presented.

Then the Modelica diff algorithm will be evaluated based on how it performs for common operations in the OMEdit GUI. Both quality and performance will be considered. Operations in OMEdit are mostly single edits followed by running the Modelica diff algorithm. This is what the algorithm was designed for.

Recently, support for conversion scripts was added to OpenModelica. This works by converting the internal representation of a loaded library, but it is also possible to create a script to write the changes to file, and merge these files with the original ones. Conversion scripts can potentially change every line of code, so this will serve as a stress test for the Modelica diff algorithm.

3 The diff algorithm

The basis of our Modelica diff algorithm is the classical Myers (1986) text diff algorithm with some additional optimizations based on ideas by Butler (2009) and Google (2019).

The ideas used from Butler (2009) and Google (2019) stem from the fact that you can easily check if the two compared sequences have a common prefix or suffix. Myers (1986) scales with the sum of the sizes of the two inputs, and if there are only changes local to a part of the file trimming away a common prefix and/or suffix will significantly improve performance of the algorithm. However, these optimizations do not improve performance if there are changes all over the file. In our algorithm, these checks also ignore whitespace (so unparsed text will be considered equal).[3]

The implementation in OpenModelica is a generic implementation because our diff algorithm is not based on text (lexer tokens, etc).[4] Instead of using a text diff, the full algorithm is performing a diff on concrete syntax trees. In order to start the diff algorithm, the inputs of both files go through a lexer and a hand-written recursive-descent parser which both preserve comments and whitespace. The output of the parser is a tree where nodes also contain whitespace and comments belonging to this subtree of the code. Where the Modelica grammar has nodes that can be given a name (such as classes, elements, or named modifications), this node is labelled by the parser. Modelica models are not allowed to define the same name twice in the same scope, making these labels unique.[5] Tichy (1984) considers blocks of text as units and moved them together instead of as in Myers (1986) where each modified unit of text that is moved is considered one move. However, as the algroithm presented below is a tree diff algorithm it is more similar to for example Matsumoto, Higo, and Kusumoto (2019) than Tichy (1984), as it works on units already divided into blocks using domain knowledge of Modelica and compares these instead of trying to create blocks from text.

The tree diff implemented in OpenModelica[6] works recursively for each node where the diff algorithm runs on the sequence of nodes in each subtree that is not equal to

[2] https://github.com/modelica/ModelicaStandardLibrary

[3] Due to the internal representation and unparsing in OpenModelica, parentheses are also considered whitespace. This is done to not have a diff when the unparsing adds or removes unnecessary parentheses.

[4] https://github.com/OpenModelica/OpenModelica/blob/master/OMCompiler/Compiler/Util/DiffAlgorithm.mo

[5] The labels are only unique for valid Modelica models. The quality of the diff is decreased when performed on invalid models.

[6] https://github.com/OpenModelica/OpenModelica/blob/master/OMCompiler/Compiler/Parsers/SimpleModelicaParser.mo

the corresponding subtree in the other sequence. When there is only 1 change in the entire sequence, most of the file will be kept the same without any possibility of adding whitespace in the wrong place. However, the algorithm is no longer scaling as $O(ND)$ since we may potentially perform this operation at each depth of the tree.

The tree diff algorithm has additional optimizations performed on the result returned by the generic diff algorithm:

1. Move operations are detected by looking only for nodes with the same label in order to improve performance. If there is a node deleted and added with identical contents, the merged result contains the text of sequence A in the position that it was moved to in sequence B. This is also performed for comments, trying to not move them.

2. Changes to whitespace are ignored unless they are needed to separate two tokens in the merged text.

3. Indentation is preserved to match the previous line in the original.

4. Nodes that are not equal to some node in the other sequence are compared to each other (this is a recursive algorithm). If the labels match, those nodes are compared to each other. If there is only one node remaining, it may have been renamed and is compared. The algorithm does not consider multiple renamed nodes at the same time and will fallback to resetting the formatting of the nodes in this case.

4 Evaluation

The diff algorithm has been extensively tested and improved since it was introduced in OpenModelica back in 2015. The first version used Myers (1986) as token-based diff, but this took several hours to perform a diff on 300 kB large files due to many whitespace changes[7].

There are two aspects of the evaluation: quality and performance. This section presents results for both of these metrics. The computer used for the performance measurements uses an AMD Ryzen 5900X CPU and has 32 GB RAM.

Listings 1 and 2 merge into the same content as Listing 1 in 633 µs. Some of the largest example models in the standard library with a diagram where components can be moved around are `ComparisonPullInStroke` and `BatchPlant_StandardWater`[8].

For `BatchPlant_StandardWater`, one component was moved in the OMEdit GUI as shown in Figure 1, causing an update to a component and two connections.

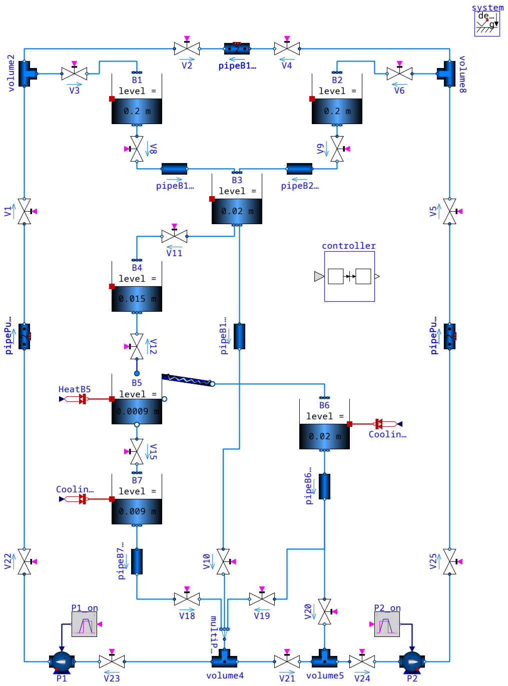


Figure 1. The diagram for BatchPlant_StandardWater. Component B2 is moved slightly down to the right in the example described in the text.

For performance reasons, OMEdit performed all 3 updates first and then called the Modelica diff algorithm, resulting in Listing 3. This causes the connections to be indented slightly incorrectly due to two adjacent nodes being updated at the same time. Note that connections are slightly more complicated to handle than components since they do not have a name and the diff algorithm does not detect that these were existing connections that were updated. When OMEdit modifies the component, the visible and rotation modifiers are added because the compiler does not keep track of which values are defaults and which had explicit modifiers. This causes the formatting of those annotation to be affected as well. The rest of the file remains the same as before, which makes the diff readable. The diff algorithm takes 0.55 s to run, which explains why OMEdit tries to perform a few edits at the same time. To illustrate that the updates look nicer, consider Listing 4 where only one connection was updated. The time it takes is slightly lower than performing 3 diffs at the same time (0.37 s), but not 3 times lower since the whole file needs to be parsed and the common prefix/suffix optimization saves some time here.

For `ComparisonPullInStroke`, a connection was added to see how the Modelica diff algorithm handles added connections. The new connection was added and changed to red color and the updated diagram can be seen in Figure 2. Listing 5 shows that the connection is added at the correct indentation level, with the same formatting as OpenModelica's default for added annotations. The diff algorithm runs faster (0.08 s) than for the larger `BatchPlant_StandardWater` file.

[7] `https://github.com/OpenModelica/OpenModelica/commit/dc2d3ef0465e`

[8] Sizes were calculated based on OpenModelica's unparsing of the example and not the size of the file it is stored in. `BatchPlant_StandardWater` is not the only class stored in the file it is defined in.

```
@@ -292,8 +292,7 @@
         diameter=0.011,
         height=0)},
       stiffCharacteristicForEmptyPort=false)
-                         annotation (Placement(transformation(extent={{50,180},
-              {90,220}})));
+                         annotation (Placement(visible = true, transformation(extent = {{98,
↪   124}, {138, 164}}, rotation = 0)));
     Modelica.Fluid.Examples.AST_BatchPlant.BaseClasses.TankWithTopPorts B3(
       redeclare package Medium = BatchMedium,
       height=0.5,
@@ -546,10 +545,10 @@
         points={{-130,220},{-120,220},{-120,230},{-90,230},{-90,221}}, color={0,127,255}));
     connect(volume8.port_3, V6.port_a) annotation (Line(
         points={{150,220},{130,220}}, color={0,127,255}));
-    connect(V6.port_b, B2.topPorts[1]) annotation (Line(
-        points={{110,220},{100,220},{100,230},{70,230},{70,221}}, color={0,127,255}));
-    connect(B2.ports[1], V9.port_a) annotation (Line(
-        points={{70,179},{70,175},{70,175},{70,170}}, color={0,127,255}));
+  connect(V6.port_b, B2.topPorts[1]) annotation(
+      Line(points = {{110, 220}, {118, 220}, {118, 165}}, color = {0, 127, 255}));
+  connect(B2.ports[1], V9.port_a) annotation(
+      Line(points = {{118, 123}, {118, 174.5}, {70, 174.5}, {70, 170}}, color = {0, 127, 255}));
     connect(V9.port_b, pipeB2B3.port_a) annotation (Line(
         points={{70,150},{70,144},{50,144}}, color={0,127,255}));
     connect(pipeB2B3.port_b, B3.topPorts[2]) annotation (Line(
```

Listing 3. `BatchPlant_StandardWater` where component B2 was moved, the connections to it updated, and the Modelica diff merged the changes. The text is a regular unified text diff of the files (since the whole file is 100 kB large).

```
@@ -547,7 +547,7 @@
     connect(volume8.port_3, V6.port_a) annotation (Line(
         points={{150,220},{130,220}}, color={0,127,255}));
     connect(V6.port_b, B2.topPorts[1]) annotation (Line(
-        points={{110,220},{100,220},{100,230},{70,230},{70,221}}, color={0,127,255}));
+        points={{110,220},{118, 220},{118, 165}}, color={0,127,255}));
     connect(B2.ports[1], V9.port_a) annotation (Line(
         points={{70,179},{70,175},{70,175},{70,170}}, color={0,127,255}));
     connect(V9.port_b, pipeB2B3.port_a) annotation (Line(
```

Listing 4. `BatchPlant_StandardWater` where only one connection was updated and the Modelica diff merged the changes. The text is a regular unified text diff of the files (since the whole file is 100 kB large). Note that there are fewer whitespace changes than in Listing 3 (OMEdit also removed some lines when re-routing the connection).

```
@@ -407,6 +407,8 @@
                                                color={0,0,255}));
   connect(simpleSolenoid.flange, simpleLoad.flange_a)
     annotation (Line(points={{0,-50},{20,-50}}, color={0,127,0}));
+  connect(simpleLoad.flange_b, advancedLoad.flange_b) annotation(
+    Line(points = {{40, -50}, {66, -50}, {66, 30}, {40, 30}}, color = {170, 0, 0}));
   annotation (experiment(StopTime=0.05, Tolerance=1e-007), Documentation(
         info="<html>
 <p>
```

Listing 5. `ComparisonPullInStroke` where a connection was added, its color changed, and the Modelica diff merged the changes. The text is a regular unified text diff of the files (since the whole file is 30 kB large).

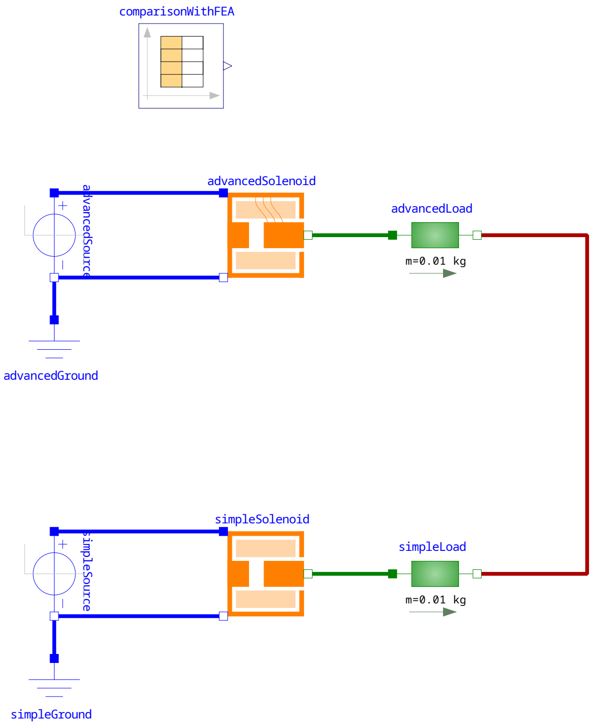
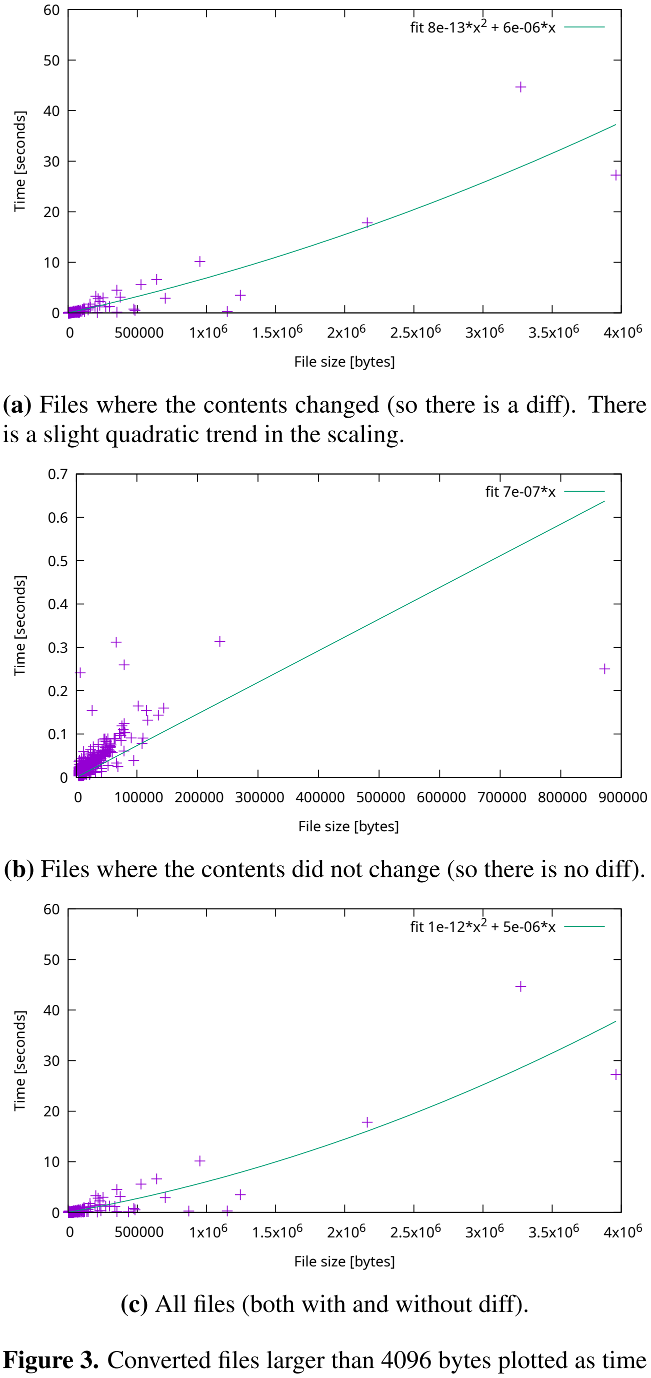


Figure 2. The diagram for `ComparisonPullInStroke`. The connection in red color was added and the examples try to merge the text before this connection with the text after the update.

The diff algorithm was also tested on libraries that had the MSL 4.0.0 conversion script applied to them.[9]. Not all files will have been updated, and the diff algorithm failed on a few files. Figures 3a and 3b show a curve fitting on the sets of files where there was a diff and where there was no diff detected after merging, comparing size to how long the diff operation takes. For smaller files, the scaling is linear (around 8x higher for files with a diff than those without). As files grow larger, they seem to have quadratic scaling. Note that the number of edits are not known for these files. If you consider all files in the same set (Figure 3c), the scaling is quadratic as the files where the content changed will dominate the overall times.

The conversion script test has also been grouped by the library that was converted and a summary can be seen in Table 1. One library that has few modified files, runs fast, and produces a very good diff[10] is the BioChem library. This is because the BioChem library mostly uses its own units based on the SI units from the standard library. The Buildings library is a much larger library, with many changed files. Since Buildings is split into over 3000 files, it does not use much memory although it takes almost a minute to complete. The quality of the diffs in Buildings



(a) Files where the contents changed (so there is a diff). There is a slight quadratic trend in the scaling.

(b) Files where the contents did not change (so there is no diff).

(c) All files (both with and without diff).

Figure 3. Converted files larger than 4096 bytes plotted as time it takes for the diff algorithm to handle problems of a given file size.

[9] A copy of the text diff compared to the original is available at `https://gist.github.com/sjoelund/b7574f7aaf052500b0835f14e4b25d95`

[10] https://github.com/OpenModelica/BioChem/commit/517a9962647

goes from great (Listing 6) to reasonable (Listing 7) with no very bad results. ScalableTestSuite and Physiomodel both have a single very large file in the library (and some smaller ones), which causes a lot of memory to be needed. Despite the large file, ScalableTestSuite produces a very nice diff whereas Physiomodel has whitespace changes in a lot of places. For the files where the diff algorithm failed, OpenModelica's unparsing of the internal form is used instead. This changes too many lines to be able to tell what semantic changes were actually performed.

5 Discussion

The method is limited to programmatically modified text. If a regular text editor is used and the diff algorithm is used to merge the changes, all the manual whitespace changes are lost. This means it cannot be used as an enforced hook in your source code management system since you would never be able to fix broken whitespace.

Handling diffs in connections is difficult because the connections do not have names. This means that subtrees are only compared if a single connection was updated. In practice, this does not seem to affect components in the conversion script.

There is another limitation in that parenthesis are considered whitespace – the unparsing of OpenModelica would need to be updated to preserve parenthesis where added manually (and to not output parentheses in other places either). This particular limitation sometimes causes the merging to fail with catastrophic results. These merge failures are detected by a sanity check that verifies that the semantics before and after merging are the same. Removing a parenthesis or moving parenthesis to the wrong location is thus detected. Not all of the failures may be due to this limitation of the diff algorithm – earlier versions of the algorithm also failed because the parser did not handle the full Modelica grammar.

The unparsing of text is also assumed to preserve formatting of real numbers. In OpenModelica, the parser keeps the text instead of transforming the value into floating point in order to produce something easier to perform the diff on. If another Modelica tool would read `1000000.0` and output `1e6`, the diff algorithm would assume this is a desired change.

If a Modelica tool would reorder for example modifications in annotations, this would cause the merged code to also move the modifications around (possibly with some changes to indentation and whitespace).

The algorithm has difficulty with performance and unintended edits when there are many changes at the same time. In order to improve performance and quality of the diff, Modelica libraries should be split into multiple files as this vastly improves responsiveness of the OMEdit GUI when moving objects around. This effect is also seen when applying conversion scripts to libraries where libraries with a small number of files have a lower quality in the diffs produced by the diff algorithm. The diff algorithm could easily be extended to run in parallel for libraries split into multiple files, further improving performance at the cost of memory usage. Splitting the library into multiple files also has a positive impact on load performance since it is trivial to parse multiple files in parallel[11].

Continuously improving the diff algorithm has fixed a lot of similar problems in the past, and with a bit more finetuning for things like indentation the algorithm could become even better.

The diff algorithm has not been evaluated on output from other Modelica tools, but a reasonable way to do so would be to run the diff algorithm on git commits in some of these libraries.

6 Conclusion

The diff algorithm was intended to be used in an interactive GUI with single changes between each modification and it works well for this use-case. When the diff algorithm is used on larger changes such as applying a conversion script, the quality of the diffs goes down and it works much better when the library is divided into many smaller files. The good news is that even when the indentation is changed, it is usually feasible to manually correct these since there are only local changes (`git diff` will not show you irrelevant edits). The time spent on making sure there are only small changes saves time for the code reviewer. Compared to existing tools that change formatting of the whole file even when not making any change, it is a big step in the right direction.

Given the existing diff algorithm, the recommendation for development of Modelica libraries would be to split the library into more files since you get a faster response from the user interface for each edit as well as fewer unintended changes. This workflow is something that should work well in most cases.

After fixing the remaining bugs and tuning the diff algorithm, it should be able to run the conversion script on all the tested libraries.

Detecting which connections belong together could be resolved by making the unparsing include additional information to both files that is then removed from the merged file. This might allow tools to perform more edits at the same time and still preserve formatting within each connection. Alternatives include looking for the names of the ports that are connected (but this might break if a connection is moved between ports), or perhaps by having named connections in the Modelica specification.

Changes that improve the quality of diffs after applying the conversion script would also improve the tool when running on files modified by other Modelica tools (for example if pull requests in Github need to be cleaned up to see what changes are actually proposed).

[11] Loading files in parallel was implemented in OpenModelica back in 2014 and typical speedup is 5x for 8 threads with the garbage collection as bottleneck. Loading a single large file vs. multiple files in a single thread has no impact on performance in our experience.

Table 1. The performance of running the conversion script. The exact libraries converted are either the latest release or from a master branch on Github (this has little relevance on the results, but only 1 copy of each library was used). Shown is the total file sizes of all files in the library, the largest file size (to see if libraries not split into smaller files are more problematic to handle), how many files were modified, how many the diff algorithm failed on, and how many files there were in the library. The time it takes to run the conversion script and the total time to run the Modelica diff algorithm on all files in the library as well as the maximum heap size for these phases are also provided. If the maximum heap size for the diff algorithm is the same as for the conversion script, it means that the memory reclaimed by the garbage collector was enough to run the diff algorithm.

Library	Size	Max size	Diffs	Fail	Files	Conversion		Diff	
AdvancedNoise	340 kB	39 kB	12	0	107	0.2 s	24 MB	0.7 s	24 MB
AixLib	12,585 kB	89 kB	972	3	2,731	5.0 s	490 MB	36.4 s	491 MB
BioChem	564 kB	231 kB	3	0	134	0.6 s	58 MB	1.1 s	107 MB
BuildSysPro	7,110 kB	79 kB	871	12	1,948	3.2 s	234 MB	22.5 s	235 MB
BuildingSystems	7,574 kB	132 kB	676	2	2,089	3.2 s	315 MB	22.7 s	316 MB
Buildings	13,947 kB	472 kB	1,117	0	3,341	5.8 s	474 MB	39.4 s	475 MB
ConPNlib	50 kB	6 kB	1	0	37	0.1 s	18 MB	0.2 s	18 MB
ElectricalEnergyStorage	330 kB	330 kB	1	1	1	0.2 s	33 MB	1.2 s	156 MB
ExternalMemoryLib	28 kB	28 kB	1	0	1	0.1 s	25 MB	0.1 s	25 MB
FCSys	1,121 kB	292 kB	7	0	16	0.9 s	58 MB	2.8 s	123 MB
FastBuildings	181 kB	4 kB	8	0	131	0.2 s	24 MB	0.7 s	24 MB
HanserModelica	372 kB	14 kB	58	0	137	0.2 s	33 MB	2.0 s	33 MB
HelmholtzMedia	477 kB	73 kB	24	0	242	0.5 s	33 MB	1.4 s	44 MB
IBPSA	4,901 kB	67 kB	494	0	1,382	2.4 s	186 MB	13.8 s	186 MB
IdealizedContact	625 kB	625 kB	1	0	1	0.2 s	44 MB	6.6 s	188 MB
IndustrialControlSystems	717 kB	19 kB	11	0	241	0.3 s	44 MB	1.5 s	44 MB
KeyWordIO	58 kB	7 kB	7	0	38	0.1 s	24 MB	0.2 s	24 MB
LibRAS	255 kB	14 kB	36	2	80	0.2 s	33 MB	1.1 s	33 MB
MEV	75 kB	75 kB	1	1	1	0.1 s	18 MB	0.2 s	44 MB
ModelicaByExample	292 kB	6 kB	54	0	355	0.7 s	33 MB	1.7 s	33 MB
Modelica_DeviceDrivers	634 kB	67 kB	30	0	192	0.7 s	44 MB	2.0 s	58 MB
Modelica_Noise	290 kB	23 kB	12	0	101	0.2 s	24 MB	0.6 s	24 MB
Modelica_Synchronous	827 kB	246 kB	7	0	9	0.4 s	44 MB	7.2 s	107 MB
NcDataReader2	12 kB	2 kB	1	0	12	0.1 s	18 MB	0.0 s	18 MB
ObjectStab	243 kB	44 kB	22	0	159	0.3 s	33 MB	0.9 s	44 MB
OpenHydraulics	530 kB	26 kB	30	0	162	0.2 s	44 MB	1.7 s	44 MB
OpenIPSL	1,294 kB	30 kB	52	0	402	0.6 s	90 MB	3.9 s	90 MB
PNlib	816 kB	76 kB	5	0	224	0.3 s	44 MB	1.6 s	59 MB
PhotoVoltaics	271 kB	21 kB	35	0	118	0.2 s	24 MB	1.2 s	33 MB
PhotoVoltaics_TGM	100 kB	7 kB	20	0	20	0.1 s	18 MB	0.5 s	18 MB
Physiolibrary	886 kB	265 kB	7	0	10	1.0 s	44 MB	4.7 s	123 MB
Physiomodel	3,417 kB	3,196 kB	2	0	4	0.9 s	186 MB	44.8 s	991 MB
PowerGrids	643 kB	26 kB	17	0	202	0.3 s	44 MB	1.6 s	44 MB
PowerSystems	1,973 kB	109 kB	5	0	104	1.6 s	90 MB	3.7 s	123 MB
ScalableTestSuite	6,186 kB	3,869 kB	14	0	22	0.8 s	254 MB	45.8 s	1,002 MB
SiemensPower	400 kB	16 kB	95	0	169	0.3 s	33 MB	1.0 s	33 MB
SolarTherm	1,161 kB	62 kB	264	2	534	1.2 s	74 MB	4.7 s	74 MB
Spot	1,944 kB	115 kB	6	0	90	1.4 s	90 MB	3.9 s	107 MB
SystemDynamics	1,216 kB	1,216 kB	1	0	1	0.3 s	74 MB	3.5 s	396 MB
ThermalSeparation	4,642 kB	1,123 kB	134	4	533	3.1 s	138 MB	14.7 s	270 MB
ThermoPower	2,502 kB	930 kB	8	1	10	1.2 s	106 MB	19.8 s	350 MB
ThermoSysPro	4,588 kB	343 kB	594	0	980	1.9 s	186 MB	24.1 s	300 MB
iPSL	3,068 kB	852 kB	41	0	534	0.7 s	106 MB	5.7 s	122 MB

```
@@ -3,7 +3,7 @@
   "Controller for single zone VAV system"
   extends Modelica.Blocks.Icons.Block;

-  parameter Modelica.SIunits.Temperature TSupChi_nominal
+  parameter Modelica.Units.SI.Temperature TSupChi_nominal
     "Design value for chiller leaving water temperature";
   parameter Real minAirFlo(
     final min=0,
@@ -11,10 +11,10 @@
     final unit="1")
     "Minimum airflow fraction of system"
     annotation(Dialog(group="Setpoints"));
-  parameter Modelica.SIunits.DimensionlessRatio minOAFra
+  parameter Modelica.Units.SI.DimensionlessRatio minOAFra
     "Minimum outdoor air fraction of system"
     annotation(Dialog(group="Setpoints"));
-  parameter Modelica.SIunits.Temperature TSetSupAir
+  parameter Modelica.Units.SI.Temperature TSetSupAir
     "Cooling supply air temperature setpoint"
     annotation(Dialog(group="Setpoints"));
   parameter Buildings.Controls.OBC.CDL.Types.SimpleController controllerTypeHea=
```

Listing 6. An example of a typical diff in Buildings, which looks good.

```
@@ -5,16 +5,16 @@

   Modelica.Blocks.Sources.Sine mixAirTem(
     amplitude=7.5,
-    freqHz=1/86400,
+f     =1/86400,
     offset=20 + 273.15) "Mixed air temperature"
     annotation (Placement(transformation(extent={{-100,80},{-80,100}})));
   Modelica.Blocks.Sources.Sine retAirTem(
     amplitude=10,
-    freqHz=1/86400,
+f     =1/86400,
     offset=21 + 273.15) "Return air temperature"
     annotation (Placement(transformation(extent={{-100,-70},{-80,-50}})));
   Modelica.Blocks.Sources.Sine outAirTem(
-    freqHz=1/86400,
+    f =1/86400,
     amplitude=6,
     offset=18 + 273.15) "Measured outdoor air temperature"
     annotation (Placement(transformation(extent={{-100,-40},{-80,-20}})));
@@ -51,7 +51,7 @@
     annotation (Placement(transformation(extent={{-100,-10},{-80,10}})));
   Modelica.Blocks.Sources.Sine supAirTem(
     amplitude=7,
-    freqHz=1/86400,
+f     =1/86400,
     offset=13 + 273.15) "Supply air temperature"
     annotation (Placement(transformation(extent={{-100,-100},{-80,-80}})));
 equation
```

Listing 7. An example of an uncommon diff in Buildings. The results are reasonable, with only local changes. However, the indentation has been changed except where `freqHz` was the first modification.

Future work includes not considering parentheses whitespace, which requires changing OpenModelica to never add or remove parentheses in its internal representation. This change would resolve most issues where files failed to merge. Implementing this change would make the algorithm only work in tools that preserved parentheses in its unparsing, making the diff algorithm only work well with OpenModelica.

Given this knowledge, it can be concluded that the given approach will not work to merge any arbitrary changes of Modelica code. However, the approach should work well when it is integrated with a tool that preserves as much of the original structure as possible (including positions of parentheses, the exact string representation of floating point numbers, as well as the order of modifications and connections).

There are alternative approaches that would make library development easier, such as Modelica tools making edits directly in the concrete syntax tree. However, this approach requires a much more targeted approach and needs to be considered at an early stage of development. Another approach would be to use the same version of a Modelica tool for all edits of Modelica code, or a formatter that enforces consistent formatting before making a commit to the source code management system. This approach makes it harder to make manual changes to the source code since the tool may automatically revert the changes as they do not conform to its formatting rules.

Acknowledgments

This research was funded by Vinnova in the ITEA EMPHYSIS project (grant number 2017-05121). Support from the Swedish Government has also been received through the ELLIIT project and through SSF in the LARGEDYN project (grant number ITM17-0154).

References

Butler, Nicholas (2009). *Investigating Myers' diff algorithm: Part 1 of 2*. URL: https://www.codeproject.com/Articles/42279/Investigating-Myers-diff-algorithm-Part-1-of-2 (visited on 2021-04-21).

Fritzson, Peter et al. (2020). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Google (2019). *Diff Match Patch*. URL: https://code.google.com/p/google-diff-match-patch/ (visited on 2021-04-21).

Hashimoto, Masatomo and Akira Mori (2008). "Diff/TS: A Tool for Fine-Grained Structural Change Analysis". In: *2008 15th Working Conference on Reverse Engineering*, pp. 279–288. DOI: 10.1109/WCRE.2008.44.

Matsumoto, Junnosuke, Yoshiki Higo, and Shinji Kusumoto (2019). "Beyond GumTree: A Hybrid Approach to Generate Edit Scripts". In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pp. 550–554. DOI: 10.1109/MSR.2019.00082.

Modelica Association (2017-04). *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.

Modelica Association (2021-02). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.5*. Tech. rep. Linköping: Modelica Association. URL: https://specification.modelica.org/maint/3.5/MLS.html.

Myers, Eugene (1986). "An $O(ND)$ difference algorithm and its variations". In: *Algorithmica* 1, pp. 251–266. DOI: 10.1007/BF01840446.

Tichy, Walter (1984-11). "The String-to-String Correction Problem with Block Moves". In: *ACM Trans. Comput. Syst.* 2.4, pp. 309–321. ISSN: 0734-2071. DOI: 10.1145/357401.357404.

Numerically Robust Six-Equation Two-Phase Flow Model for Stationary and Moving Systems in Modelica

Johannes Brunnemann[1] Ales Vojacek[1] Thomas Koch[1]

[1]XRG Simulation GmbH, Germany, {brunnemann,vojacek,koch}@xrg-simulation.de

Abstract

We present a physics based Modelica finite volume flow model that separately balances vapour and liquid phase. By using extensive state variables and a special mass flow regularisation, the model can cope with the possible vanishing or emerging of a phase in a numerically robust way. Although at prototype stage, the model already exhibits all required capabilities. These are demonstrated in feature testers and in a model of a natural convection driven cooling cycle operating under external acceleration forces.

Keywords: two phase flow, six equation model, evaporating and cooling cycle, natural convection, moving systems, ClaRa library

1 Introduction

For the modelling of two phase flows, the assumption of a homogeneous spatial mixture of liquid and vapour phase is widely used in Modelica. Both phases are taken in thermal and mechanical equilibrium (equal temperature and static pressure) and form a lumped mass flow. However these assumptions are not always applicable, in particular in situations where vapour and liquid phase are expected to move independently. Within the NAKULEK[1] project options for passive cooling of power electronics in aircrafts have been investigated regarding their dimensioning and reliability under aircraft flight conditions. In these cooling circuits liquid coolant evaporates at the hot electronic equipment. The vapour then releases its heat in a condenser, see figure 7. The flow of the coolant is solely driven by natural convection. However sufficient heat removal has to be ensured at any time during operation of the aircraft. Hence the effect of external acceleration forces due to flight manoeuvres on the coolant flow has to be analysed, in particular rotations of the cooling circuit. These may lead to induced liquid flows, shifting vapour and liquid volume fractions at different spatial positions in the circuit. Experimental studies have been conducted by the project partners TUHH (Albertsen and Schmitz 2019) and ZAL (Quaium and Kuhn 2020), that additionally employ phase change material at the evaporator in order to buffer heat flow peaks.

Based on the ClaRa library (ClaRa Development Team 2021) a Modelica library containing supplementary system models of the test facilities was created by the project partner XRG Simulation GmbH (Brunnemann 2020).

This paper introduces a central element of that library: a detailed two phase flow model, based on a finite volume realisation of the so called 6-equation approach. It provides balance equations for mass, energy and momentum separately for vapour and liquid phase (hence 6-equation model) and considers dynamic external acceleration while allowing counter-directional movement of the phases. Although the model is at prototype stage it already exhibits all desired capabilities, as demonstrated in section 3.

2 Model Development

The 6-equation model approach is well established in the literature (Whalley 1987; Sokolichin 2003; Brennen 2005; Ghiaasiaan 2008) and realised in several power plant simulators, e.g. APROS (Hänninen and Ylijoki 2008). In Modelica, two phase flows are mostly treated as homogeneous flows with common balance equations for both phases (3-equation model) (as e.g. in (Francke 2014) or the Modelica Standard library). Sometimes these homogeneous models are extended by phenomenological models for interphase velocity difference (slip) or heat transfer and/or friction, e.g. in (Hoppe, Gottelt, and Wischhusen 2017).

An alternative treatment is the moving boundary approach (Jensen and Tummescheit 2002; Bonilla et al. 2012), where the spatial regions of single-phase and two phase are computed dynamically, but the two-phase region is still modelled as a homogeneous model.

In (Bauer 1999) a an advanced evaporator model was presented, with a common energy balance of the phases but (optional) separate momentum balance. This model already demonstrated the advantage of extended balancing, however numerical problems occurred at vanishing vapour phase.

Separate balancing of vapour and liquid phase, doubles the number of balance equations per control volume. Beside the doubling the equations, such an extension introduces a substantial number of additional flows in the balance equations, due to interaction of the co-existing phases. An overview is given in Table 1.

2.1 Limitations of Specific Quantity Approach

For realizing a 6-equation model based on finite control volumes in Modelica one has to consider that the control

[1] NAKULEK - Natural Circulation driven Cooling of Power Electronics (German: Naturumlaufkühlung für Leistungselektronik).

DOI 10.3384/ecp21181539

Table 1. Flows considered in homogeneous 3-equation flow model compared to 6-equation model. Simple doubling for both phases is denoted by "(liq+vap)", interaction between phases is denoted by "(liq↔vap)". Additional flows of the 6-equation model are marked in blue.

Flow	*3 equation*	*6 equation*	
Mass flows	convective	convective	(liq+vap)
		phase change	(liq↔vap)
Enthalpy flows	convective	convective	(liq+vap)
		phase change	(liq↔vap)
Heat flows	to wall	to wall	(liq+vap)
		interphase	(liq↔vap)
Momentum flows	static p	static p	(liq+vap)
		water level	(liq+vap)
	dynamic p	dynamic p	(liq+vap)
		phase change	(liq↔vap)
	wall friction	wall friction	(liq+vap)
		interphase slip	(liq↔vap)
	gravity	gravity	(liq+vap)
		external acceleration	

volume V_ℓ and hence the volume fraction $\varepsilon_\ell = V_\ell / V_{\text{tot}}$ of a particular phase $\ell \in \{\text{liq,vap}\}$ varies with time. Only the total volume $V_{\text{tot}} = V_{\text{vap}} + V_{\text{liq}}$ is constant. Moreover for single phase flow, the other phase is totally absent. For an equation based Modelica model this implies that the time evolution for states of that particular phase ℓ becomes meaningless in the limit $\varepsilon_\ell \to 0$. This issue has been addressed e.g. in (Jensen and Tummescheit 2002; Bonilla et al. 2012) where time evolution of states of the vanishing phase are mapped onto those of the other phase as dummy equation. For this mapping the form of the balance equations has to be modified: they need to be "switched over" for volume fractions close to zero but also "switched back" to the original zone physics if the volume of that zone exceeds a certain lower bound.

The ClaRa library (ClaRa Development Team 2021) features pipe models using a homogeneous 3-equation finite volume approach, where a pipe flow is discretised along flow direction into a one dimensional so called energy grid consisting of N_{cv} control volumes (energy cells). In each energy cell specific enthalpy h and static pressure p are chosen as states. Moreover flow velocity w is balanced on a staggered flow grid consisting of $N_{cv} + 1$ flow cells, see Figure 1 with (for the 3-equation model assumed) unified vapour/liquid control volumes. Time evolution for pressure is derived from the mass balance via Equation 11 and Equation 12 by using the fact that $V = const$ for the homogeneous 3-equation-approach.

While the 3-equation model assumes thermal and mechanical equilibrium (equal temperatures and static pressure) as well as spatial homogeneity of the phases, the 6-equation model only assumes mechanical equilibrium (equal static pressures). From that we created (as a first attempt) a 6-equation model with state variables $h_{vap}, h_{liq}, w_{vap}, w_{liq}, p, \varepsilon_{vap}$ and tried to cope with vanishing phases according to the "switching" of (Bonilla et al. 2012). However it turned out that the according modification of the balance equations causes numerical stability issues. In particular the "switching" procedure appears to be problematic, as all balance equations are numerically coupled. Additionally the "switch back" to physical time evolution for an emerging phase turns out to be hard to define consistently. The definition of the state derivatives for $h_{vap/liq}, w_{vap/liq}$ becomes meaningless if $mass_{vap/liq} \to 0$. Moreover the volume fraction $\varepsilon_{vap/liq}$ is directly involved into computation of friction pressure loss and heat transfer through computation of contact surfaces. If ε_{vap} is a state, then numerically it may happen that $0 \leq \varepsilon \leq 1$ can be violated by numerical precision. This in turn produces numerical instabilities, e.g. diverging heat flows. A rethinking of these issues revealed the following insights:

1. It is easier to regulate flows in a conservation law than to regulate the actual form of that law.
2. In the context of vanishing masses and dynamic control volumes we should refrain from using specific quantities as states. Rather we should only balance "countable" (extensive) quantities.
3. Static Pressure p and volume fraction ε_{vap} should not be used as states.

2.2 From Specific to Absolute Quantities

Consequently we decided to base the 6-equation model on absolute quantities, rather then specific quantities. See Table 2 for a comparison. This means, that the specific quantities are not states. The time evolution of the system does not depend on the behaviour of the specific quantities. They are just used as algebraic functions in order to define the flows of enthalpy, momentum and volume as well as inputs to the media model:

Table 2. Absolute and specific quantities. For completeness particle number N and particle weight M are given in order to illustrate the 'extensive' nature of the absolute quantities. For $V = const$, Equation 11 and Equation 12 can be used in order to define a time evolution for pressure instead of mass.

Quantity	*absolute*	*specific*
particle number	N	
mass	m	$M = m/N$
internal energy	U	$u = U/m$
enthalpy	H	$h = H/m$
momentum	I	$w = I/m$
volume	V	$v = V/m$

$$H_{flow} = h \cdot m_{flow} \quad I_{flow} = w \cdot m_{flow} \quad V_{flow} = v \cdot m_{flow} \quad (1)$$

Notice that these definition are independent of time varying cross sectional area $A_{\oslash,\ell}$ or volume V_ℓ: mass and absolute quantity are independent of volume as well as the mass flow rate $m_{flow,\ell}$, which can be computed from momentum I_ℓ according to

$$m_{flow,\ell} = w_\ell \cdot \rho_\ell \cdot A_{\oslash,\ell} = \frac{I_\ell}{\delta x} \quad . \tag{2}$$

Here we have used $A_{\oslash,\ell} = \varepsilon_\ell \cdot A_\oslash$ and the fact that overall control volume $V_{\mathrm{tot}} = A_\oslash \cdot \delta\bar{x}$ ($\delta\bar{x}$ is the discretization length of the flow grid) is constant in time.

2.3 Alternative State Selection

We will now work out the transition to new extensive state variables in detail:

$$\{p_\ell, h_\ell, w_\ell\} \to \{m_\ell, U_\ell, H_\ell, I_\ell\} \tag{3}$$

Avoidance of specific quantities introduces one more state on the right hand side. In order to avoid p, ε as states, we use absolute Enthalpy H_ℓ in addition to internal energy U_ℓ. The model will be set up on a staggered grid according to (Figure 1). Circles inside the control volumes in Figure 1 represent state locations. Small "j" denotes flow cell labels. Capital "J" denotes energy cell labels. The volume fraction ε_{vap} is variable in each control volume. Each quantity "x" naturally defined on one of the grids can be defined on the other grid by suitable interpolation. The interpolated quantity "$\bar{x}$" is marked by an overline.

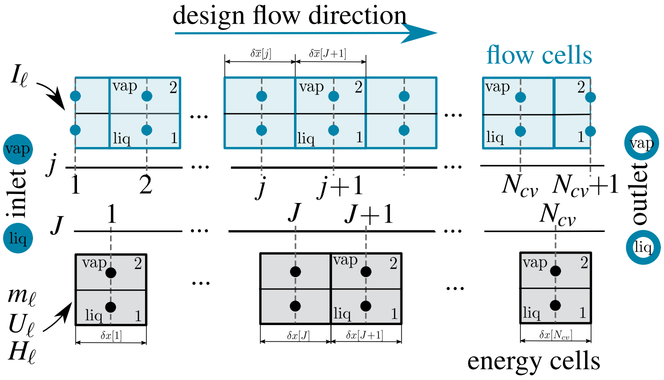


Figure 1. Staggered grid used in model with inlet and outlet connectors for each phase $\ell \in \{1,2\} \equiv \{\mathrm{liq},\mathrm{vap}\}$. Top line: flow cells with momentum balance (I_ℓ). Bottom line: energy cells with mass (m_ℓ) and energy (U_ℓ, H_ℓ) balance.

We will denote the species index by ℓ, where $\ell = 1 \Leftrightarrow$ liq, $\ell = 2 \Leftrightarrow$ vap. Moreover we introduce the sign

$$\sigma_\ell = \begin{cases} 1 \text{ if } \ell = 1 \\ -1 \text{ if } \ell = 2 \end{cases}$$

With this convention we have for the total volume $V_{\mathrm{tot}} = \mathrm{const} = V_1 + V_2$ and define the volume fractions

$$\varepsilon_1 = \frac{V_1}{V_{\mathrm{tot}}} = 1 - \varepsilon \qquad \varepsilon_2 = \frac{V_2}{V_{\mathrm{tot}}} = \varepsilon \quad , \tag{4}$$

which implies

$$\frac{dV_2}{dt} = -\frac{dV_1}{dt} = V_{\mathrm{tot}} \frac{d\varepsilon}{dt} \tag{5}$$

We assume mechanical equilibrium between the phases:

$$p = p_1 = p_2 \qquad \frac{dp}{dt} = \frac{dp_1}{dt} = \frac{dp_2}{dt} \tag{6}$$

2.3.1 Mass Balance

It is straight forward to show for the mass m_ℓ:

$$\begin{aligned} \frac{d}{dt} m_\ell[J] \quad &= \quad m_{flow,\ell}[j] - m_{flow,\ell}[j+1] \\ &\quad + \sigma_\ell \, m^{(cond)}_{flow}[J] - \sigma_\ell \, m^{(evap)}_{flow}[J] \end{aligned} \tag{7}$$

Mass flows $m_{flow,\ell}[j]$ are computed from momentum I_ℓ according to Equation 44. The phase change mass flows $m^{(cond)}_{flow}, m^{(evap)}_{flow}$ are computed according to section 2.6.3.

2.3.2 Energy Balance

For the internal energy U_ℓ we have:

$$\begin{aligned} \frac{d}{dt} U_\ell[J] \quad &= \quad H_{flow,\ell}[j] - H_{flow,\ell}[j+1] \\ &\quad + Q^{(\ell \to int)}_{flow}[J] + Q^{(\ell \to wall)}_{flow}[J] \\ &\quad + \sigma_\ell \, H^{(cond)}_{flow}[J] - \sigma_\ell \, H^{(evap)}_{flow}[J] \\ &\quad - p[J] \, \frac{dV_\ell[J]}{dt} \end{aligned} \tag{8}$$

Note that we have neglected kinetic and potential energy. For large flow velocities of considerable masses or for flows along vertical pipes these terms can be added. The convective enthalpy flows $H_{flow,\ell}[j]$ are obtained according to Equation 40. The conductive heat flows $Q^{(\ell \to int)}_{flow}[J], Q^{(\ell \to wall)}_{flow}[J]$ are described in section 2.6.1. The enthalpy flows due to phase change are computed as

$$H^{(cond)}_{flow}[J] = m^{(cond)}_{flow}[J] \cdot h^{(bub)}[i] \tag{9}$$

$$H^{(evap)}_{flow}[J] = m^{(evap)}_{flow}[J] \cdot h^{(dew)}[i] \tag{10}$$

where $h^{(bub)}, h^{(dew)}$ denote bubble / dew specific enthalpy. This assumes that the phase change enthalpy difference $\Delta h^{(evap)} = h^{(dew)} - h^{(bub)}$, stays inside the outgoing phase: condensation heat $Q^{(cond)}_{flow} = \Delta h^{(evap)} m^{(cond)}_{flow}$ stays inside the vapour phase and conversely evaporation heat $Q^{(evap)}_{flow} = \Delta h^{(evap)} m^{(evap)}_{flow}$ is taken from the liquid phase. In this way phase change is numerically stabilized, as liquid phase is more cooled and vapour phase more heated by phase change.

The last term on the right hand side of Equation 8 denotes possible expansion work, as the control volume of each phase ℓ is variable (Skogestad 2009).

2.3.3 Enthalpy Time Evolution

In the following we will leave out the ℓ-index for simplicity, but re-introduce it at the end. From the definition of mass $m = \rho V$ it follows for the density ρ in a time-dependent control volume V:

$$\frac{d\rho}{dt} = \frac{1}{V}\left\{\frac{dm}{dt} - \rho\frac{dV}{dt}\right\} \tag{11}$$

But from $\rho = \rho(p,h)$ it also holds that

$$\begin{aligned}\frac{d\rho}{dt} &= \underbrace{\left.\frac{\partial\rho}{\partial p}\right|_h}\frac{dp}{dt} + \underbrace{\left.\frac{\partial\rho}{\partial h}\right|_p}\frac{dh}{dt}\\ &= \quad A\ \frac{dp}{dt} \ + \ B\ \frac{dh}{dt}\end{aligned} \tag{12}$$

This can be written as:

$$\frac{dp}{dt} = \frac{1}{A}\left\{\frac{d\rho}{dt} - B\frac{dh}{dt}\right\} \tag{13}$$

From $U = H - pV$ it follows that

$$\frac{dH}{dt} = \frac{dU}{dt} + \frac{dp}{dt}V + p\frac{dV}{dt} \tag{14}$$

Moreover

$$\frac{dh}{dt} = \frac{d\left(\frac{H}{m}\right)}{dt} = \frac{1}{m}\left\{\frac{dU}{dt} + \frac{dp}{dt}V + p\frac{dV}{dt} - \frac{dm}{dt}h\right\} \tag{15}$$

Here we have used Equation 14. Now we plug Equation 11 and Equation 15 into Equation 13. After some algebraic manipulations we arrive at:

$$\frac{dp}{dt} = \frac{1}{V}\frac{1}{X}\left\{Y\frac{dm}{dt} - B\frac{dU}{dt} - Z\frac{dV}{dt}\right\}, \tag{16}$$

where we have introduced the shorthands:

$$X = \rho A + B \quad Y = \rho + Bh \quad Z = \rho^2 + Bp$$

Now we can plug this into Equation 14 in order to replace the dp/dt-term. After some manipulation this gives:

$$\frac{dH}{dt} = \rho\frac{A}{X}\frac{dU}{dt} + \frac{Y}{X}\frac{dm}{dt} - \rho\frac{W}{X}\frac{dV}{dt} \quad . \tag{17}$$

Here we have used $W = \rho - pA$.

Application to liq-vap-system Now we re-introduce the species indices and set $\ell = 1 \Leftrightarrow$ liq, $\ell = 2 \Leftrightarrow$ vap. Using Equation 16 we thus get:

$$\frac{dp_1}{dt} = \frac{1}{V_1}\frac{1}{X_1}\left\{Y_1\frac{dm_1}{dt} - B_1\frac{dU_1}{dt} - Z_1\frac{dV_1}{dt}\right\} \tag{18}$$

$$\frac{dp_2}{dt} = \frac{1}{V_2}\frac{1}{X_2}\left\{Y_2\frac{dm_2}{dt} - B_2\frac{dU_2}{dt} - Z_2\frac{dV_2}{dt}\right\} \tag{19}$$

Now we subtract Equation 19 from Equation 18. Using Equation 6 and Equation 4, Equation 5 we can express:

$$\begin{aligned}V_{\text{tot}}\frac{d\varepsilon}{dt} &= -\frac{\varepsilon_2X_2Y_1}{Q_{12}}\frac{dm_1}{dt} + \frac{\varepsilon_2X_2B_1}{Q_{12}}\frac{dU_1}{dt}\\ &\quad + \frac{\varepsilon_1X_1Y_2}{Q_{12}}\frac{dm_2}{dt} - \frac{\varepsilon_1X_1B_2}{Q_{12}}\frac{dU_2}{dt}\end{aligned} \tag{20}$$

where $Q_{12} := \varepsilon_1X_1Z_2 + \varepsilon_2X_2Z_1$. Writing out Equation 17 for both phases gives:

$$\begin{aligned}\frac{dH_1}{dt} &= \rho_1\frac{A_1}{X_1}\frac{dU_1}{dt} + \frac{Y_1}{X_1}\frac{dm_1}{dt} + \rho_1\frac{W_1}{X_1}V_{\text{tot}}\frac{d\varepsilon}{dt}\\ \frac{dH_2}{dt} &= \rho_2\frac{A_2}{X_2}\frac{dU_2}{dt} + \frac{Y_2}{X_2}\frac{dm_2}{dt} - \rho_2\frac{W_2}{X_2}V_{\text{tot}}\frac{d\varepsilon}{dt}\end{aligned}$$

Now we plug in Equation 20 in order to replace the $d\varepsilon/dt$-term and can finally write for the time derivatives of the total enthalpies :

$$\left(\tfrac{d}{dt}H_1, \tfrac{d}{dt}H_2\right) = \left(\tfrac{d}{dt}m_1, \tfrac{d}{dt}U_1, \tfrac{d}{dt}m_2, \tfrac{d}{dt}U_2\right)\ \mathfrak{A} \tag{21}$$

with the matrix $\mathfrak{A}$ given as

$$\mathfrak{A} = \begin{pmatrix} \frac{Y_1}{X_1} - \rho_1\frac{W_1}{X_1}\frac{\varepsilon_2X_2Y_1}{Q_{12}} & \rho_2\frac{W_2}{X_2}\frac{\varepsilon_2X_2Y_1}{Q_{12}} \\ \rho_1\frac{A_1}{X_1} + \rho_1\frac{W_1}{X_1}\frac{\varepsilon_2X_2B_1}{Q_{12}} & -\rho_2\frac{W_2}{X_2}\frac{\varepsilon_2X_2B_1}{Q_{12}} \\ \rho_1\frac{W_1}{X_1}\frac{\varepsilon_1X_1Y_2}{Q_{12}} & \frac{Y_2}{X_2} - \rho_2\frac{W_2}{X_2}\frac{\varepsilon_1X_1Y_2}{Q_{12}} \\ -\rho_1\frac{W_1}{X_1}\frac{\varepsilon_1X_1B_2}{Q_{12}} & \rho_2\frac{A_2}{X_2} + \rho_2\frac{W_2}{X_2}\frac{\varepsilon_1X_1B_2}{Q_{12}} \end{pmatrix}$$

2.3.4 Momentum Balance

As is the case for mass and energy balance, we use absolute momentum I_ℓ as state variable, due to the time dependence of the control volume V_ℓ. Also the cross sectional flow area $\overline{A}_{\oslash,\ell}$ varies with time. Therefore we use momentum flows denoted by I_{flow}, that is we balance forces instead of force area densities (pressure drops).

$$\begin{aligned}\frac{d}{dt}I_\ell[j] &= I^{(stat)}_{flow,\ell}[j] + I^{(grav)}_{flow,\ell}[j] - I^{(wall)}_{flow,\ell}[j] + \sigma_\ell\, I^{(int)}_{flow,\ell}[j]\\ &\quad + I^{(adv)}_{flow,\ell}[J-1] - I^{(adv)}_{flow,\ell}[J] + I^{(WL)}_{flow,\ell}[J-1] - I^{(WL)}_{flow,\ell}[J]\\ &\quad + \sigma_\ell\, I^{(cond)}_{flow,\ell}[j] - \sigma_\ell\, I^{(evap)}_{flow,\ell}[j]\end{aligned} \tag{22}$$

Static pressure force. We have

$$\frac{I^{(stat)}_{flow,\ell}[j]}{A_\oslash} = \Delta p[j]\overline{\varepsilon}_\ell[j] + \mathsf{C}_{\text{supp}}\frac{d}{dt}\Big(\Delta p[j]\overline{\varepsilon}_\ell[j]\Big)\frac{\tau_{\text{pass}}[j]}{2} \tag{23}$$

where $\Delta p[j] = (p[J-1] - p[J])$ is the static pressure difference. $A_\oslash$ denotes the overall cross sectional area of the pipe. The second term on the right hand side can be activated via $\mathsf{C}_{\text{supp}} \in \{0,1\}$ in order to suppress numerical high frequency oscillations with $\tau_{\text{pass}}[j] = \delta\overline{x}[j]/w_{\text{sound}}[j]$ the passing time of a sound wave through the length $\delta\overline{x}[j]$ flow control volume $V[j]$.

DOI 10.3384/ecp21181539

Force due to gravity and external acceleration. The model considers acceleration $\vec{g}_{grav}$ due to gravity as well as dynamic external accelerations $\vec{g}_{ext}$ due to movement of the pipe. The resulting overall acceleration vector $\vec{g}$ is given by $\vec{g} = \vec{g}_{grav} + \vec{g}_{ext}$. Now consider the unit vetor $\vec{e}_x$ pointing into design flow direction of the pipe. It is given by $\vec{e}_x = (\vec{r}_{out} - \vec{r}_{in})/|\vec{r}_{out} - \vec{r}_{in}|$, where $\vec{r}_{out}, \vec{r}_{in}$ are the position vectors of the outlet, inlet frame connector (see section 2.7). Now we can decompose $\vec{g}$ into

$$\vec{g} = \vec{g}_{\parallel} + \vec{g}_{\perp} \tag{24}$$

where $\vec{g}_{\parallel} = \langle \vec{g}, \vec{e}_x \rangle \vec{e}_x$, $\langle \cdot, \cdot \rangle$ denotes the scalar product. Using $g_{\parallel} = |\vec{g}_{\parallel}|$ we can write for the overall force $I^{(grav)}_{flow,\ell}[j]$ induced by gravity and acceleration:

$$I^{(grav)}_{flow,\ell}[j] = g_{\parallel}\ \overline{m}_{\ell}[j] \tag{25}$$

The remainder $\vec{g}_{\perp}$ in Equation 24 is perpendicular to design flow direction and is given by $\vec{g}_{\perp} = \vec{g} - \vec{g}_{\parallel}$.

Water level force. Using $g_{\perp} = |\vec{g}_{\perp}|$ of Equation 24, we can write

$$I^{(WL)}_{flow,\ell}[J] = \frac{m_{\ell}[J]}{\delta x[J]}\ \mathtt{WL}_{\ell}[J]\ g_{\perp} \tag{26}$$

WL is the water level height, computed from the spatial separation model (see section 2.6.4). At clear spatial separation of the phases, WL causes 'acceleration' pressure $p^{(acc)}_{\ell} = \rho_{\ell} \cdot g_{\perp} \cdot \mathtt{WL}_{\ell}$ (mostly) inside the liquid. This causes an effective static pressure $p^{(eff)}_{stat,\ell} = p_{stat} + p^{(acc)}_{\ell}$. Since static pressure acts isotropically, this in turn results in $I^{(WL)}_{flow,\ell}$ along flow direction (compare to Equation 23). Here we use $\rho_{\ell} = \frac{m_{\ell}}{V_{\ell}} = \frac{m_{\ell}}{\delta x A_{\oslash} \varepsilon_{\ell}}$. Multiplying $p^{(acc)}_{\ell}$ by the phase cross sectional area $A_{\oslash}\varepsilon_{\ell}$ Equation 26 is obtained.

Wall Friction and Interphase Friction. are denoted by $I^{(wall)}_{flow,\ell}[j]$ and $I^{(int)}_{flow,\ell}[j]$, see section 2.6.2.

Force due to advection (dynamic pressure). Based on the usual formulation of the advective force,

$$I^{(adv)}_{flow,\ell} = w_{\ell} \frac{I_{\ell}}{\delta x} = w_{\ell}^2 \cdot \rho_{\ell} \cdot A_{\oslash,\ell}\ , \tag{27}$$

and seeing how the mass flow and flow velocity are computed from the momentum state (Equation 2), we may write:

$$I^{(adv)}_{flow,\ell}[J] = \begin{cases} w_{\ell}[j] m_{flow,\ell}[j] & \text{if } \overline{w}_{\ell}[J] > 0 \\ w_{\ell}[j+1] m_{flow,\ell}[j+1] & \textit{else} \end{cases} \tag{28}$$

In this formulation we avoid the time varying cross sectional area and density.

Phase Change Forces Beside mass and enthalpy transfer, phase change also causes momentum transfer between the phases.

$$I^{(evap)}_{flow,\ell}[j] \quad = \quad \overline{m}^{(evap)}_{flow}[j]\ w_1[j] \tag{29}$$

$$I^{(cond)}_{flow,\ell}[j] \quad = \quad \overline{m}^{(cond)}_{flow}[j]\ w_2[j] \tag{30}$$

Here the interpolated phase change mass flows are computed according to Equation 45.

2.4 Regularization of the Media Data in Case of a Vanishing Phase

We consider two VLE-media, one for the vapor phase and one for the liquid phase, that take pressure p and specific enthalpy p as inputs. Moreover we use a VLE-object taking the overall homogeneous specific enthalpy

$$h_{\text{hom}} = \frac{H_1 + H_2}{m_1 + m_2}$$

Due to the mechanical equilibrium assumption $p = p_1 = p_2$ it holds for bubble specific enthalpy that $h_1^{(\text{bub})} = h_2^{(\text{bub})} = h_{\text{hom}}^{(\text{bub})} = h^{(\text{bub})}$ and for dew specific enthalpy $h_1^{(\text{dew})} = h_2^{(\text{dew})} = h_{\text{hom}}^{(\text{dew})} = h^{(\text{dew})}$. We use the actual specify enthalpy

$$h_{\ell} = \frac{H_{\ell}}{\max\left(m_{reg}, m_{\ell}\right)} \tag{31}$$

as auxiliary quantity in order to define the regularized specific quantities

$$\begin{aligned} h_1^{(reg)} &= \min\left(h^{(bub)}, h_1\right) \\ h_2^{(reg)} &= \max\left(h^{(dew)}, h_2\right) \end{aligned} \tag{32}$$

Here we have introduced a regulator m_{reg} for vanishing phase. The thus defined specific enthalpies $h_1^{(reg)}, h_2^{(reg)}$ are taken as input to the VLE-media objects together with static pressure p. Note that Equation 32 allows for a short time that specific enthalpy h_{ℓ} of a phase ℓ enters two phase region. However due to evaporation and condensations mass flows of Equation 54 the phase will return to pure phase after a while. The suggested construction avoids numerical issues due to heavily varying media data inside two-phase-region.

2.4.1 Pressure and Volume Fraction

The introduction of the new state variables now allows to define static pressure p and volume fraction ε_{ℓ} in terms of the new states. For $\ell = \{1,2\} = \{liq, vap\}$ we introduce the overall enthalpy $H_{\text{tot}} = H_1 + H_2$ and the total inner energy $U_{\text{tot}} = U_1 + U_2$ as well as the volume fraction $\varepsilon_1 = \varepsilon_{liq} = 1 - \varepsilon$, $\varepsilon_{vap} = \varepsilon_2 = \varepsilon$. Now we use the definitions $H_{\ell} = U_{\ell} + pV_{\ell}$ and $H_{\text{tot}} = U_{\text{tot}} + pV_{\text{tot}}$ and Equation 6 in order to write down

$$p = \frac{H_{\text{tot}} - U_{\text{tot}}}{V_{\text{tot}}} \qquad \varepsilon_{\ell} = \frac{H_{\ell} - U_{\ell}}{H_{\text{tot}} - U_{\text{tot}}} \tag{33}$$

In turn this also allows to express the time derivatives

$$\frac{d}{dt} p = \frac{1}{V_{\text{tot}}} \left(\frac{d}{dt} H_{\text{tot}} - \frac{d}{dt} U_{\text{tot}} \right) \tag{34}$$

Differentiating ε_{ℓ} with respect to time and simplifying the obtained expressions one gets for the time derivative of the volume fraction:

$$-\frac{d}{dt}\varepsilon_1 = \frac{d}{dt}\varepsilon_2 = \frac{1}{V_{\text{tot}}} \cdot \frac{\varepsilon_1 X_1 R_2 - \varepsilon_2 X_2 R_1}{Q_{12}^{(reg)}}\ , \tag{35}$$

where $R_\ell = Y_\ell \frac{dm_\ell}{dt} - B_\ell \frac{dU_\ell}{dt}$ and

$$Q_{12}^{(reg)} = \begin{cases} \varepsilon_{reg} & \text{if } |Q12| < \varepsilon_{reg} \\ Q_{12} & \text{else} \end{cases}$$

2.5 Consistent Interpolation of Half Spaced Quantities on Staggered Grid

Here we give a brief description, how quantities defined on either energy or flow cell grid Figure 1 can be consistently defined on the other grid by interpolation.

Volume

$$\overline{V}_\ell[j] = \frac{V_\ell[J-1] + V_\ell[J]}{2} \qquad \overline{V}_{\text{tot}}[j] = \overline{V}_1[j] + \overline{V}_2[j] \quad (36)$$

Cross Sectional Area

$$\overline{A}_{\oslash,\ell}[j] = \overline{A}_{\oslash}[j]\ \overline{\varepsilon}_\ell[j] \qquad \overline{A}_{\oslash}[j] = \frac{\overline{V}_{\text{tot}}[j]}{\delta\overline{x}[j]} \quad (37)$$

Length of Volume Element

$$\delta\overline{x}[j] \quad = \quad \frac{\delta x[J-1] + \delta x[J]}{2} \quad (38)$$

Volume Fraction

$$\overline{\varepsilon}_\ell[j] = \frac{\overline{V}_\ell[j]}{\overline{V}_{\text{tot}}[j]} = \frac{V_{\text{tot}}[J-1]}{2\overline{V}_{\text{tot}}[j]}\ \varepsilon_\ell[J-1] + \frac{V_{\text{tot}}[J]}{2\overline{V}_{\text{tot}}[j]}\ \varepsilon_\ell[J] \quad (39)$$

Mass of Flow Cells While the previous quantities are defined in a straight forward way, the mass $m_\ell[j]$ of a flow cell needs some additional considerations. In a homogeneous 3-equation model one would choose $\overline{m}_\ell[j] = (m_\ell[J-1] + m_\ell[J])/2$. However this is not consistent with the possible vanishing of a particular phase. To see this recall Equation 2, that expresses the mass flow $m_{flow,\ell}$ in terms of the momentum I_ℓ: To see this, consider two neighbour energy cells, with $\varepsilon_2[J-1] = 0$ and $\varepsilon_2[J] > 0$. Clearly, there cannot be vapour mass flow from $[J-1] \rightarrow [J]$, as $m_2[J-1] = 0$. Only vapour mass flow in opposite direction $[J] \rightarrow [J-1]$ may occur. And the model shall account for this. If we 'count' mass as a state, then the pure mass flows carry that quantity, similarly to e.g. enthalpy flows

$$H_{flow,\ell}[j] = m_{flow,\ell}[j]\overline{h}_\ell[j] \quad , \quad (40)$$

where the mass flow carries specific enthalpy $\overline{h}[j]$. In the latter case one often uses an upstream scheme in order to define specific enthalpy $\overline{h}[j]$ at the center of a flow cell, that is

$$\overline{h}_\ell[j] = \begin{cases} h_\ell[J-1] & \text{if } m_{flow,\ell}[j] > 0 \\ h_\ell[J] & \text{if } m_{flow,\ell}[j] < 0 \end{cases} \quad (41)$$

Accordingly we may use an upstream scheme for the mass $\overline{m}_\ell[j]$ of a flow cell:

$$\overline{m}_\ell^{(up)}[j] = \varsigma_\ell^{(I)}[j] \cdot m_\ell[J-1] + \left(1 - \varsigma_\ell^{(I)}[j]\right) \cdot m_\ell[J]$$

with $\varsigma_\ell^{(I)}[j] = \text{sm}(I_{reg}, 0, I_\ell[j])$, where sm$(\cdot)$ denotes the stepSmoother function contained in `Modelica.Fluid.Dissipation.Utilities.` I_{reg} is a regulator. Then we set

$$\overline{m}_\ell[j] = \max\left(m_{reg}, \overline{m}_\ell^{(up)}[j]\right) \quad , \quad (42)$$

with m_{reg} a regulator. At present, the approach still uses absolute boundaries for the regulators. In principle, these should be scaled according to a characteristical smallest number (such as smallest length or volume) of the system. This way, the approach will be robust for varying system sizes.

Flow Velocities The flow velocities can be well defined:

$$w_\ell[j] = \max\left(-w_s[j], \min\left(w_s[j], \frac{I_\ell[j]}{m_\ell[j]}\right)\right) \quad , \quad (43)$$

where we limit the flow velocity to the speed of sound w_s in order to avoid unrealistic flow velocities in the limit of small masses m_{reg}, which may lead to unwanted frictional momentum flows. By construction our model assumes subsonic flow speeds. At energy cell flow locations we use the averaged momentum

$$\overline{w}_\ell[J] = \max\left(-w_s[J], \min\left(w_s[J], \frac{I_\ell[j] + I_\ell[j+1]}{m_\ell[J]}\right)\right)$$

Mass Flows are then written as

$$m_{flow,\ell}[j] = \varsigma_\ell^{(\overline{m})}[j] \cdot \frac{I_\ell[j]}{\delta\overline{x}[j]} \quad , \quad (44)$$

where $\varsigma_\ell^{(\overline{m})}[j] = \text{sm}\left(m_{max}, m_{min}, \overline{m}_\ell[j]\right)$ and m_{max}, m_{min} are regularization parameters. This construction ensures that outgoing mass flow of a particular phase goes to *numeric* zero if the mass of that phase inside the control volume approaches zero. In particular no mass of a phase ℓ can be extracted from a control volume with $\varepsilon_\ell = 0$. On the other side mass can be easily injected from control volumes with $\varepsilon_\ell > 0$ into control volumes with $\varepsilon_\ell = 0$. This becomes especially important in situations, where e.g. vapour is injected from the outside into a pipe entirely filled with liquid.

Phase Change Mass Flows at momentum state location are computed as the sum of the phase change mass flows of the two adjacent energy half cells:

$$\overline{m}_{flow}^{(evap)}[j] = \frac{m_{flow}^{(evap)}[J-1] + m_{flow}^{(evap)}[J]}{2} \quad (45)$$

$$\overline{m}_{flow}^{(cond)}[j] = \frac{m_{flow}^{(cond)}[J-1] + m_{flow}^{(cond)}[J]}{2} \quad (46)$$

2.6 Replaceable Models

2.6.1 Heat Transfer

For each position J we have for the heat flows:

$$Q_{flow}^{(\ell\rightarrow int)} \quad = \quad \varsigma_\ell^{(m)} \cdot \alpha_{int} \cdot A_{12} \cdot (T_{int} - T_\ell) \quad (47)$$

$$Q_{flow}^{(\ell\rightarrow wall)} \quad = \quad \varsigma_\ell^{(m)} \cdot \alpha_{\ell,wall} \cdot A_{\ell,wall} \cdot (\texttt{heat.}T - T_\ell) \quad (48)$$

Here again $\varsigma_{\ell}^{(m)}[J] = \mathrm{sm}\left(m_{max}, m_{min}, m_{\ell}[J]\right)$ denotes the stepsmoother function. The mean interphase surface temperature T_{int} is computed from imposing a steady state energy balance at the phase contact surface:

$$0 = \sum_{\ell} Q_{flow}^{(\ell \to int)} \tag{49}$$

Similarly the heat flow for the heat port is computed as

$$\mathtt{heat}[J].Q_{flow} = \sum_{\ell} Q_{flow}^{(\ell \to wall)}[J] \tag{50}$$

2.6.2 Momentum Transfer

Wall + interfacial friction/ heat transfer models should give truly zero momentum/heat flow at vanishing phase. As default we use a 0-equation turbulence (mixing length) approach (VERSTEEG and MALALASEKERA 1995), which describes the effect of turbulence by an effective modification of dynamic viscosity μ_{ℓ}:

$$\mu_{\ell}[j] = \overline{\mu}_{\ell}^{(0)}[j] \cdot (1 + \mathtt{CF}_{\ell} \cdot \mathrm{Re}_{\ell}[j]); \tag{51}$$

with Reynolds number $\mathrm{Re}_{\ell}[j] = \left|w_{\ell}[j]\right| \overline{\rho}_{\ell}[j] \delta\overline{x}[j] / \overline{\mu}_{\ell}^{(0)}[j]$ and a calibration factor $\mathtt{CF}_{\ell}$. Then we can write for the friction between the phases

$$I_{flow,\ell}^{(int)}[j] = \frac{(w_2[j] - w_1[j])}{\max(l_{reg}, \Delta l_{12}[j])} \, \mu_{\ell}[j] \, A_{12}[j] \tag{52}$$

Similarly we have for the friction force between phase ℓ and the pipe walls:

$$I_{flow,\ell}^{(wall)}[j] = \frac{w_{\ell}[j]}{\max(l_{reg}, \Delta l_{\ell,wall}[j])} \, \mu_{\ell}[j] \, A_{\ell,wall}[j] \tag{53}$$

Moreover A_{12} denotes the contact area between the phases and Δl_{12} the mean distance between the center of the phase control volumes. Similarly $A_{\ell,wall}$ denotes the contact surface area between phase ℓ and pipe wall and $\Delta l_{\ell,wall}$ denotes the mean distance between phase control volume and pipe wall. At present stage, we assume ideal phase separation to derive these quantaties, as described in subsubsection 2.6.4. A flow regime model that computes A_{12} and Δl_{12} from an effective flow pattern (e.g., ideally separated, homogeneous mixture) will be subject to future work. Also, a more sophisticated turbulence model could be implemented. From our experience Equation 51 ensures that wall and interphase friction play together in a numerically stable way.

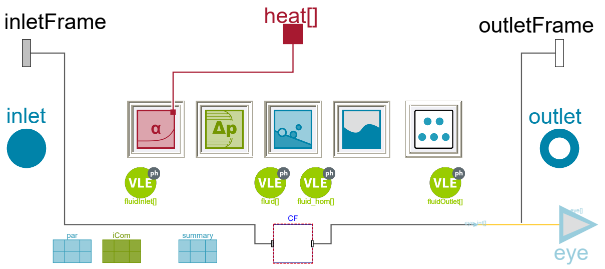


Figure 2. Diagram layer with connectors and replaceable models for heat transfer, pressure drop, phase change, spatial distribution and geometry.

2.6.3 Phase Change Models

The evaporation and condensation massflows are the massflows from the liquid to the vapour phase control volume and vice versa. They are considered to be proportional to the respective volume and, therefore, the available mass. Moreover, they scale with the steam quality and come to a halt, if the outgoing phase vanishes:

$$\begin{aligned} m_{flow}^{(evap)}[J] &= \frac{\varsigma_1^{(\varepsilon)}[J]}{\tau_{evap}} \max\left(0, \frac{h_1[J]}{h^{(bub)}[J]} - 1\right) \max\left(0, m_1[J]\right) \\ m_{flow}^{(cond)}[J] &= \frac{\varsigma_2^{(\varepsilon)}[J]}{\tau_{cond}} \max\left(0, 1 - \frac{h_2[J]}{h^{(dew)}[J]}\right) \max\left(0, m_2[J]\right) \end{aligned} \tag{54}$$

Here $\varsigma_{\ell}^{(\varepsilon)}[J] = \mathrm{sm}\left(\varepsilon_{max}, \varepsilon_{min}, \varepsilon_{\ell}[J]\right)$ and $\varepsilon_{max}, \varepsilon_{min}$ are regularization parameters. Moreover we have time constants τ_{evap}, τ_{cond}. They can be thought of average time it takes for bubbles to exit from liquid to vapour phase and mean time it takes water drops coming from vapour phase in order to enter liquid phase. Larger constants mean a slower phase change mass flow, while smaller time constants would imply very dynamic phase changes. The ansatz could be improved by a flow regime depending boiling model, considering the bubble formation, mean bubble diameter and travel distance. The time constants should also be affected by the contact area of the phases.

2.6.4 Spatial Separation

This model computes certain average contact areas and distances as well as water level. So far a simple model is implemented, assuming ideal phase separation and a circular pipe cross section.

$$\begin{aligned} A_{12}[j] &= 4 \cdot \max(0, \varepsilon_1[j] \cdot \varepsilon_2[j] \cdot A_{12}^{(0)}[j]) \\ A_{\ell,wall}[j] &= \max(0, \overline{\varepsilon_{\ell}}[j] \cdot \overline{A}_{wall}[j]) \\ \Delta l_{12}[j] &= 4 \cdot \max(d_{\oslash}/100, \overline{\varepsilon}_1[j] \cdot \overline{\varepsilon}_2[j] \cdot d_{\oslash}/2) \\ \Delta l_{\ell,wall}[j]] &= \max(d_{\oslash}/100, \overline{\varepsilon}_{\ell}[j] \cdot d_{\oslash}/2) \end{aligned} \tag{55}$$

here $d_{\oslash}$ is the pipe diameter and $A_{12}^{(0)}[j] = \delta\overline{x}[j] \cdot d_{\oslash}$ is the maximum contact surface of the phases in a horizontal cylindrical pipe at ideal separation. Water level $\mathtt{WL}_1[J]$ is computed from $d_{\oslash}$ and liquid volume fraction $\varepsilon_1[J]$ for a horizontal cylinder volume, assuming ideal separation. Consequently $\mathtt{WL}_2[J] = d_{\oslash} - \mathtt{WL}_1[J]$.

2.6.5 Geometry

The model features different geometries, smilarly to ClaRa pipes, in particular it covers pipe bundles.

2.7 Connectors

Flow connectors are build from a two element array of ClaRa flow connectors `ClaRa.Basics.Interfaces.FluidPortIn` and `ClaRa.Basics.Interfaces.FluidPortOut`, one for each phase. A vanishing phase is not problematic, since the static pressure is equal for both phases and if we ensure to have a momentum state at one side of the connector. Also in this case $h_{outflow}$ of that phase is physically not relevant and can be set to a dummy value. To connect the 6-equation model to a 3-equation component, however, one would need a suitable adapter. The adapter needs to ensure the compatibility with the flow situation: In particular a homogeneous 3-equation model cannot cope with counter flow of the phases. One `ClaRa.Basics.Interfaces.HeatPort_a` **Heat** port is attached to every control volume of the energy grid. In order to account for external acceleration each model carries **Frame** connectors from the `Modelica.Mechanics.MultiBody` package `Interfaces.Frame_a`, `Interfaces.Frame_b`. They also ensure consistency of three dimensional pipe arrangements.

3 Applications

The newly developed 6-equation model was put into feature testing (single pipe) and system testing (several pipes in system application). In this section we present three prominent examples.

3.1 Feature Tests

3.1.1 Condensation in a Tilted Pipe

This is a classic example of counter-phase flow, which is also a fair challenge for conventional CFD models: Slightly overheated steam (at 3 bars) is injected from a mass flow source into an inclined pipe from the inlet. The bottom of the pipe is connected to a vapour and liquid pressure boundary condition. The first and the last 20 m of the pipe wall are heated to 295 °C. The middle section of the pipe wall (40 m) is cooled to 5 °C, such that condensation occurs. The condensed liquid flows downward in the direction of the slope. Pressure drop due to condensation causes backflow of vapour in pipe section close to the outlet. Hence, vapour is sucked into the pipe while liquid rinses out at outlet. The selected results are presented in Figure 4. The tester demonstrates applicability of the model for heat exchangers where two phase flow occurs.

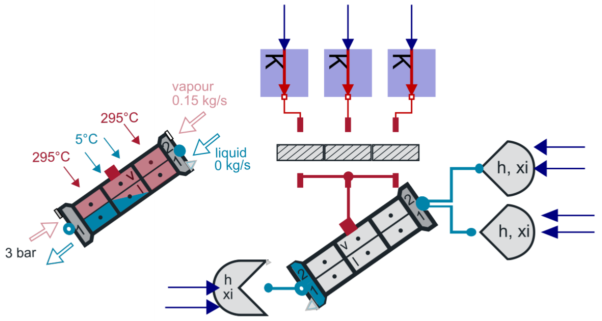


Figure 3. Condensation in the tilted pipe test model. Geometry: length L=80 m, diameter $d_\oslash$=1 m, discretisation N_{CV}=40, inclination Δz=20 m.

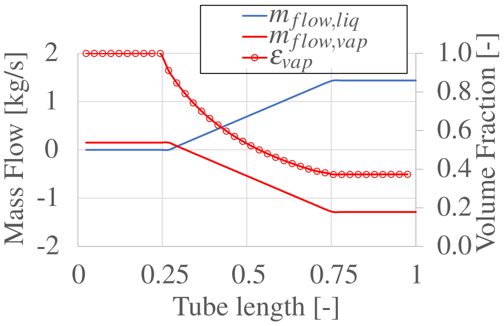


Figure 4. Resulted steady state mass flow of vapour and liquid together with volume fraction of vapour along the tilted pipe during condensation scenario.

Not only the stationary hardware, but also moving devices e.g. in vehicles, aircrafts or ships can be simulated.

3.1.2 Rotation Test

A horizontal tube is filled half with vapour and with liquid ($\varepsilon_{liq} = \varepsilon_{vap} = 0.5$). The tube is then rotated 90°downwards and back to the to the initial position. The tube is discretized with an odd number of control volumes. The model is shown in Figure 5. The results of the simulation scenario is presented in Figure 6 where the actual rotation angle is displayed below, and above it, the volume fraction of the liquid phase at the beginning, in the middle and at the end of the tube is shown. Before the rotation, the volume fractions are all at 50%. After the rotation, there is no more liquid at the top of the tube, while the tube end is completely filled with liquid. As expected, the volume fraction settles vertically at 50% liquid and 50% vapour. When turning back to horizontal position again, a decaying wave formation is visible (enlarged area), before the liquid level settles again uniformly at 50%.

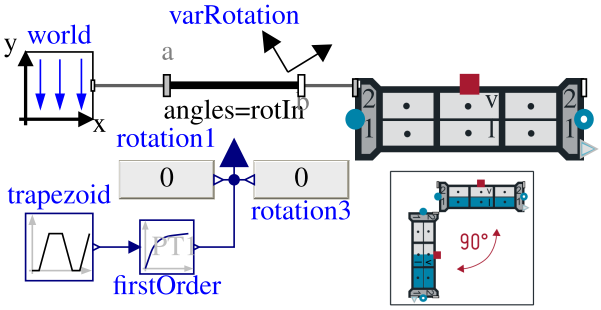


Figure 5. Rotation test model. Geometry: L=10 m, D_{in}=0.02 m, N_{CV}=41.

3.2 Aircraft Cooling Circuit

3.2.1 Test Rig Model

The project partners at TUHH and ZAL (Albertsen and Schmitz 2019; Quaium and Kuhn 2020) provided detailed information on their passive cooling cycle test rigs, as well as extensive data on the conditions and results of the measurement campaigns. The basic structure of the respective

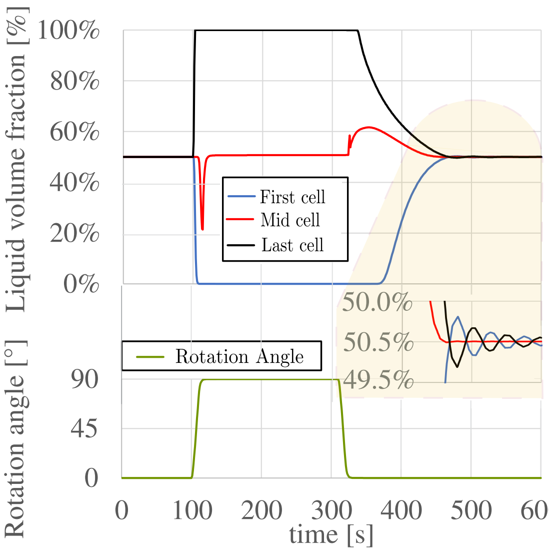


Figure 6. Resulted liquid volume fraction of the rotation test scenario.

test rigs was largely identical, the main difference being the use of a single or three parallel evaporators. In both cases, phase change material was considered as a heat load buffer at the evaporator. Figure 7 shows our test rig model within the Dymola graphical environment. The rising/-downcoming pipes and mass flow meter, as well as the evaporator and condenser models, are all based on the 6-equation flow models described above.

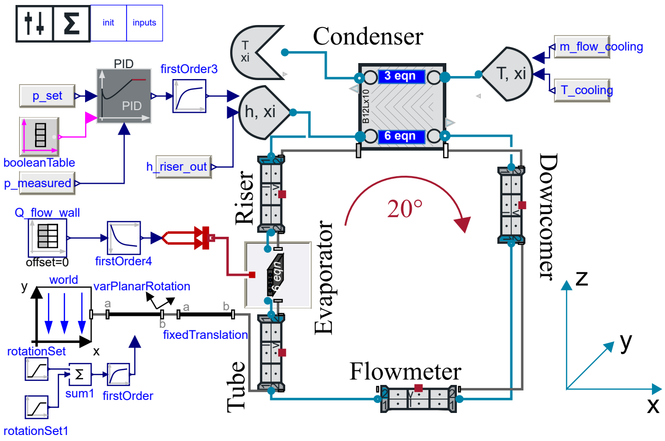


Figure 7. Diagram of measurement test rig model.

3.2.2 Heat Up and Shut Down Scenario

The first test scenario involves a sudden increase in heating power at the evaporator, followed by a sudden shutdown of the heater. Simulation results and measurement data are compared in Figure 8. The model is brought to steady state (corresponding the starting point of measured data) in several steps. Simulation is started with liquid in all the pipes and the cooling is on. After around 500 s, the heating is switched on. PID controller removes portion of flow until pressure reaches set point during operation. At around 2000 s the PID controller is disabled and the circuit is closed (self-regulating) and the system stabilizes (reaches steady state at 710 W).

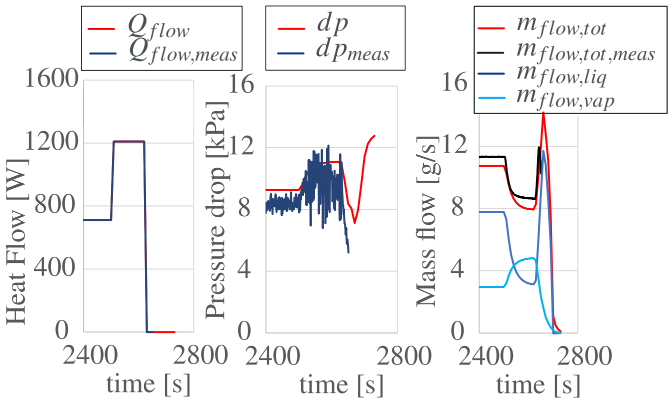


Figure 8. Heat flow (left), Pressure drop between evaporator inlet and condenser inlet (mid) and Mass flows of vapour and liquid (right), during heat up and shut down scenario.

During steady state operation at 710 W, the total mass flow through the system is around 11 g/s. Mass flow of liquid mass flow of vapour in evaporator is 8 g/s and 3 g/s respectively. During heat up to 1210 W, ε_{vap} increases. As a consequence mass flow of liquid drops to 3 g/s and mass flow of vapour increases to 5 g/s. After around 115 s (holding the new higher power level), a sudden power off (0 W) is introduced. This causes a decrease of vapour mass flow to 0 g/s and mass flow of liquid shortly increases (peaking after 30 s from shut down) as evaporator walls are still hot and pressure drop is high (caused by previous high mass flow of vapour). After reaching the peak (14 g/s), the mass flow of liquid also goes to zero, as there is no driving force (no heating). Figure 8 also shows the pressure drop between evaporator inlet and condenser outlet. There is higher pressure drop at higher heating power resulting from higher mass flow of vapour. Simulation results are in very good agreement with measured values. All steady state, heating ramp up and shut down processes were captured very well. Although measurements only provide information on the total mass flow, information on vapour/liquid mass flow can be derived (using total mass flow, pressure in the system and inlet/outlet temperature in evaporator) and energy and mass balance equations. The simulated results correspond very well to the derived values.

3.2.3 Rotation Scenario

The second scenario involves a rotation of the test rig running in steady state, reflecting a typical manoeuvre during a flight. The system is heated up by 850 W and steady state operation is achieved. At 2500 s a sudden (within 10 s) clockwise rotation by 20°around Y axis is introduced. After 300 s the system is turned back to normal position and reaches its initial steady state. Our model already captures the measured total mass flow drop during rotation qualitatively well. The quantitative deviation seems to be caused by the sudden change of spatial liquid/vapour distribution and indicates that the currently simple models for friction and spatial distribution need further elaboration.

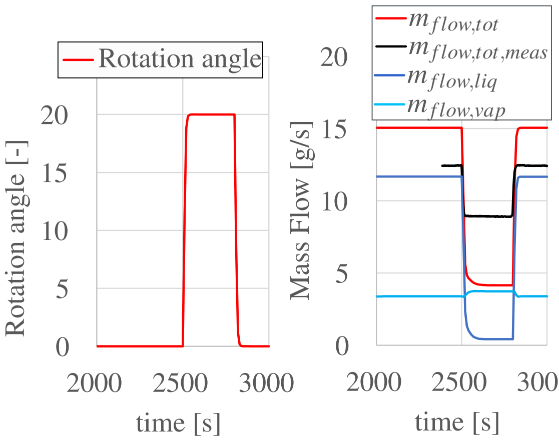


Figure 9. Rotation angle (left) and Mass flows of vapour and liquid (right) during the rotation scenario.

4 Summary & Outlook

In this paper we present a numerically robust implementation of detailed two phase model based on the six-equation approach. To achieve this we introduce an alternative set of extensive state variables from which specific quantities static pressure and volume fractions of the vapour and liquid phase can be computed algebraically. The applicability of the model is demonstrated in several testers. Currently the model is further developed to feature more detailed models for friction, spatial distribution and heat transfer, e.g. (Hoppe, Gottelt, and Wischhusen 2017). It is also straightforward to extend the model to feature multicomponent media. The model has potential application in several areas of engineering, from aerospace, automotive and naval systems design to power plants and process technologies. Typical industrial systems with two phase heat exchangers such as evaporators, boilers, steam generators and condensers would be typical example of application, especially when it comes to non-standard transient operation scenarios, e.g. start up/shut down, heat up/cool down, filling or draining of the system under investigation.

Acknowledgements

The authors would like to thank the NAKULEK research team as well as E. Makhova, T. Tumforde and J. Eiden from XRG for support and discussions. This work has been carried out within the NAKULEK research project, funded by the German Federal Ministry for Economic Affairs and Energy (grant number 20Q1519B).

Supported by:
Federal Ministry
for Economic Affairs
and Energy
on the basis of a decision
by the German Bundestag

Moreover we thank the anonymous referees for their valuable remarks.

References

Albertsen, Björn and Gerhard Schmitz (2019). *NAKULEK - Entwurf, Bau und Erprobung eines PCM-Kühlplatten Verbunds für eine Naturumlaufkuehlung von Flugzeugsystemen : Abschlussbericht*. German. Technical University Hamburg, Germany. DOI: https://doi.org/10.2314/KXP:1753985420. (Visited on 2021-04-19).

Bauer, O. (1999). "Modeling of two-Phase Flows with Modelica". MA thesis. Department of Automatic Control, Lund Institute of Technology, Sweden.

Bonilla, J. et al. (2012). "Object-Oriented Library of Switching Moving Boundary Modelsfor Two-phase Flow Evaporators and Condensers". In: *Proceedings of the 9th International Modelica Conference*, pp. 71–80.

Brennen, Christopher E. (2005). *Fundamentals of Multiphase Flows*. Cambridge University Press. ISBN: 0521 848040.

Brunnemann, Johannes (2020). *NAKULEK - Naturumlaufkühlung für Leistungselektronik : Schlussbericht*. German. XRG Simulation GmbH. DOI: https://doi.org/10.2314/KXP:1755577478. (Visited on 2021-04-19).

ClaRa Development Team, ed. (2021). *ClaRa*. Version 1.6.0. URL: https://claralib.com/.

Francke, H. (2014). "Thermo-hydraulic model of the two-phase flow in the brine circuit of a geothermal power plant". PhD thesis. Technical University Berlin,Germany.

Ghiaasiaan, S. Mostafa (2008). *Two-Phase Flow, Boiling and-Condensation. IN CONVENTIONAL ANDMINIATURE SYSTEMS*. Cambridge University Press.

Hänninen, Markku and Jukka Ylijoki (2008). *The one-dimensional separate two-phase flow model of APROS*. English. VTT Tiedotteita - Meddelanden - Research Notes 2443. VTT Technical Research Centre of Finland. 65 pp. ISBN: 978-951-38-7225-0. URL: http://www.vtt.fi/inf/pdf/tiedotteet/2008/T2443.pdf (visited on 2021-04-19).

Hoppe, T., F. Gottelt, and S. Wischhusen (2017). "Extended Modelica Model for Heat Transfer of Two-Phase Flows in Pipes Considering Various Flow Patterns". In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. Linköping University Electronic Press, Linköpings universitet, pp. 467–476.

Jensen, J.M. and H. Tummescheit (2002). "Moving Boundary Models for Dynamic Simulations of Two-Phase Flows". In: *Proceedings of the 2nd International Modelica Conference*, pp. 235–244.

Quaium, Farid and Holger Kuhn (2020). *NAKULEK - Naturumlaufkuehlung für Leistungselektronik : Abschlussbericht*. German. ZAL Center of Applied Aeronautical Research, Hamburg, Germany. DOI: https://doi.org/10.2314/KXP:1747878758. (Visited on 2021-04-19).

Skogestad, S. (2009). *Chemical and Process Engineering*. CRC Press. Chap. 11.

Sokolichin, Alexander (2003). "Mathematical modeling and numerical simulation of gas-liquid bubbly flows". Habilitation Thesis, University of Stuttgart, Germany.

VERSTEEG, H. K. and W. MALALASEKERA (1995). *An introduction to computational fluid dynamics. The finite volume method*. Longman Scientific & Technical, pp. 62–67.

Whalley, P. B. (1987). *Boiling, Condensation, and Gas-Liquid Flow*. Clarendon Press, Oxford.

Compile-Time Impulse Analysis in Modelica

Albert Benveniste[1] Benoît Caillaud[1] Mathias Malandain[1]

[1]Inria Centre de Rennes Bretagne Atlantique, University of Rennes 1, France,
{albert.benveniste,benoit.caillaud,mathias.malandain}@inria.fr

Abstract

Since its 3.3 release, Modelica offers the possibility to specify models of dynamical systems with multiple modes having different DAE-based dynamics. However, the handling of mode changes by the current Modelica tools is not satisfactory. An important difficulty is the occurrence of impulsive behavior at some mode changes, for some variables. In this paper, we propose a compile-time algorithm for identifying such impulsive behaviors and quantifying them in terms of their magnitude orders. Such algorithm can be used as an additional step of the structural analysis of Modelica models.

Keywords: multimode DAE, structural analysis, impulsive behaviors

1 Introduction

Modelica and other languages supporting object-oriented modeling of physical systems rely on the formalism of DAEs. Compilers of such languages perform sophisticated preprocessing prior to generating simulation code (Casella 2015). Index analysis and reduction (Mattsson and Soderlind 1993) is one such important processing, where selected equations are differentiated one or more times until the Jacobian matrix with respect to the leading variables (i.e., the variables of maximal differentiation degree in the system) becomes structurally regular.

Since its 3.3 release, Modelica offers the possibility of specifying *multimode dynamics,* by describing state machines with different DAE dynamics in each different state (Elmqvist et al. 2012). This feature enables describing large complex cyber-physical systems with different behaviors in different modes.

While being very valuable, this possibility has been the source of serious difficulties for non-expert users. Although many large-scale complex Modelica models are properly handled, some physically meaningful models do not give rise to correct simulation results—it is actually not difficult to construct such problematic programs, thus, chances are significant to produce such bad cases in large models. Benveniste, Caillaud, and Malandain (2020) proposes a structural analysis that is valid for multimode DAE models, both within each mode and at mode changes, illustrated in the companion paper (Benveniste, Caillaud, and Malandain 2021).

One specific problem is due to the existence, in many physical models, of impulsive behaviors for some variables. With existing tools, such models give rise to simulations collapsing at runtime. Impulsive behaviors are already a problem from a mathematical standpoint, as they do not fall within the existing concepts of solutions of a DAE system—the definition used in (Campbell and Gear 1995) assumes smoothness of the trajectories.

To cope with this issue, *distributions* were considered by some authors. To our knowledge, the most comprehensive approach was provided by Stephan Trenn. In his PhD thesis (Trenn 2009a) and his article (Trenn 2009b), he pointed out the difficulty in defining piecewise smooth distributions: several mathematically coherent definitions of the “Dirac part” of such a distribution can be considered, so that it has no intrinsic definition. This indicates that distributions are not the ultimate answer to deal with impulsive variables in multimode DAE systems. Still, Liberzon and Trenn (2012) were able to define complete solutions for a class of switched DAE systems in which each mode is in *quasi-linear form* and switching conditions are time-based, not state-based.

Another important step forward was done in (Benveniste, Caillaud, Elmqvist, et al. 2019). An interesting subclass of multimode DAE systems was identified, which possibly exhibit impulsive variables at mode changes. They extend the “quasi-linear systems” proposed by Trenn in the sense that switching conditions are no longer restricted to time-based ones, instead including state-based switching conditions. The analysis and discretization schemes proposed in (Benveniste, Caillaud, Elmqvist, et al. 2019) are mathematically sound. Building on this work, Martin Otter has developed the ModiaMath[1] tool for semi-linear multimode DAE systems. Since this work, this approach was refined and extended by the authors of this paper (Benveniste, Caillaud, and Malandain 2020), and is illustrated on examples in (Benveniste, Caillaud, and Malandain 2021).

Contribution of this paper: A complete structural analysis of multimode DAE systems was only recently proposed by the authors of this paper. In particular, this approach distinguishes between *long modes*, in which the dynamics is continuous-time and governed by a DAE system for a positive duration, and *transient modes*, which are zero duration events at which restarts can occur; note that, as a result, chattering behavior such as encountered when

[1] https://modiasim.github.io/ModiaMath.jl/stable/man/Overview.html

DOI
10.3384/ecp21181549

applying sliding mode control is not supported.

We develop here another important aspect of our approach, by focusing on impulsive behaviors. We explain this aspect on the Cup-and-Ball example, a mild variation of the popular 2D pendulum in which the straight rod is replaced by a rope. When the rope gets straight, an impulse typically occurs for the tension if an idealized model is considered. To analyze this behavior, we propose a general compile-time analysis, acting as an additional step of the multimode structural analysis presented in the companion paper (Benveniste, Caillaud, and Malandain 2021).

Since distributions fail to properly handle impulsive behaviors in general, our mathematical tool for this is *nonstandard analysis* (Robinson 1996; Cutland 1988; Lindstrøm 1988), which allows for a correct use of infinities and infinitesimals in mathematical analysis. We use this setting in two ways:

- First, we discretize the DAE dynamics in each long mode using an explicit first-order Euler scheme with an *infinitesimal* time step ∂; this provides us with an approximation of the DAE solutions up to an infinitesimal error. Infinitesimal time steps are also used to capture restarts at mode changes: the values of states in the new mode are computed, from values before the change, in one or several infinitesimal time steps.

- Second, we compute *impulse orders*, i.e., orders of magnitude of algebraic variables at mode changes, for both long and transient modes, with reference to the infinitesimal time step ∂; for example, an order of $1/\partial$ for an algebraic variable indicates that this variable is impulsive.

We develop a compile-time calculus that evaluates the impulse order of every algebraic variable, thus revealing its impulsive/non-impulsive nature. Finite impulse orders can be used to renormalize impulsive variables when implementing a numerical scheme that approximates the restart values for each state variable of the system, thus improving conditioning.

In the next section, we investigate the Cup-and-Ball example, a two-mode variation of the celebrated pendulum in Cartesian coordinates. In Section 3, we develop the impulse analysis in its generality and explain how it can be mechanized.

2 The Cup-and-Ball example

We sketch here a multimode extension of the popular example of the pendulum in Cartesian coordinates (Pantelides 1988), namely the Cup-and-Ball game illustrated by Figure 1. A ball, modeled by a point mass, is attached to one end of a rope, while the other end of the rope is fixed, to the origin of the plane in the model. The ball is subject to the unilateral constraint set by the rope, but moves freely while the distance between the ball and the origin is less than its actual length. The system is assumed closed and subject to no external interaction.

Figure 1. The Cup-and-Ball game.

2.1 The model

The considered model of the two-dimensional Cup-and-Ball game is:

$$\left\{\begin{array}{ll} 0 = x'' + \lambda x & (e_1) \\ 0 = y'' + \lambda y + g & (e_2) \\ 0 \leq L^2 - (x^2 + y^2) & (\kappa_1) \\ 0 \leq \lambda & (\kappa_2) \\ 0 = \left[L^2 - (x^2 + y^2)\right] \times \lambda & (\kappa_3) \end{array}\right. \qquad (1)$$

where the *dependent variables* are the position (x,y) of the ball in Cartesian coordinates and the rope tension λ.

The subsystem $(\kappa_1, \kappa_2, \kappa_3)$ expresses that the distance of the ball from the origin is less than or equal to L, the tension is nonnegative, and one cannot have a nonzero tension and a distance less than L at the same time. This is known as a *complementarity condition*, written as $0 \leq L^2 - (x^2 + y^2) \perp \lambda \geq 0$ in the *nonsmooth systems* literature (Acary and Brogliato 2008), and is an adequate modeling of ideal valves, diodes (Cellier and Kofman 2006, Chapter 9.10), and contact in mechanics.

Note that, not only an impulsive behavior is expected on the torques, but an other possible difficulty is present, as subsystem $(\kappa_1, \kappa_2, \kappa_3)$ of (1) leaves the impact law at mode change insufficiently specified; it could be fully elastic, fully inelastic, or in between. *We expect both of these aspects to be detected at compile time, using some kind of structural analysis.*

However, before such a structural analysis is possible, some changes are required in the model. As a matter of fact, constraints κ_1 and κ_2 are unilateral, which is not supported by Modelica and related languages. Therefore, using the technique presented in (Mattsson, Otter, and Elmqvist 1999), we redefine the graph of this complementarity condition as a parametric curve, represented by the following three equations:

$$\begin{array}{rcl} s & = & \texttt{if}\ \gamma\ \texttt{then}\ -\lambda\ \texttt{else}\ L^2 - (x^2 + y^2) \\ 0 & = & \texttt{if}\ \gamma\ \texttt{then}\ L^2 - (x^2 + y^2)\ \texttt{else}\ \lambda \\ \gamma & = & [s \leq 0] \end{array}$$

which allows us to rewrite model (1) as follows:

$$\left\{\begin{array}{rll} & 0 = x'' + \lambda x & (e_1)\\ & 0 = y'' + \lambda y + g & (e_2)\\ & \gamma = [s \leq 0] & (k_0)\\ \texttt{if}\ \gamma\ \texttt{do} & 0 = L^2 - (x^2 + y^2) & (k_1)\\ \texttt{and} & 0 = \lambda + s & (k_2)\\ \texttt{if not}\ \gamma\ \texttt{do} & 0 = \lambda & (k_3)\\ \texttt{and} & 0 = (L^2 - (x^2 + y^2)) - s & (k_4) \end{array}\right. \qquad (2)$$

We then observe that the subsystem collecting equations (k_0)–(k_4) is a logico-numerical fixpoint equation, with dependent variables x, y, λ, γ. A possible solution would consist in performing a relaxation, by iteratively updating the numerical variables based on the previous value for the guards, and then re-evaluating the guard based on the updated values of the numerical variables, hoping for a fixpoint to occur. Such fixpoint equation, however, can have zero, one, several, or infinitely many solutions. No characterization exists that could serve as a basis for a (graph-based) structural analysis. *We thus decided to refuse solving such mixed logico-numerical systems.* As a consequence, we are unable to evaluate guard γ, so that the mode the system is in cannot be determined: model (2) is rejected.

To break the fixpoint equation defining γ, we choose to restrict ourselves to guards defined by left-limits; in this example, this yields $\gamma = [s^- \leq 0]$, where $s^-(t) =_{\text{def}} \lim_{u \nearrow t} s(u)$ (the modification is highlighted in red):

$$\left\{\begin{array}{rll} & 0 = x'' + \lambda x & (e_1)\\ & 0 = y'' + \lambda y + g & (e_2)\\ & \gamma = [s^- \leq 0] & (k_0)\\ \texttt{if}\ \gamma\ \texttt{do} & 0 = L^2 - (x^2 + y^2) & (k_1)\\ \texttt{and} & 0 = \lambda + s & (k_2)\\ \texttt{if not}\ \gamma\ \texttt{do} & 0 = \lambda & (k_3)\\ \texttt{and} & 0 = (L^2 - (x^2 + y^2)) - s & (k_4) \end{array}\right. \qquad (3)$$

We are now ready to associate a structural analysis to model (3) that will be valid both in long modes with DAE dynamics, and at mode changes. To achieve this, we will replace derivatives by their corresponding forward Euler schemes, which will bring everything to a discrete progress of time (both continuous dynamics and mode changes).

To avoid introducing approximation errors, we will use an *infinitesimal* time step ∂, which is made mathematically formal by relying on *nonstandard analysis*.

2.2 Using nonstandard analysis

Nonstandard analysis (Robinson 1996; Lindstrøm 1988; Benveniste, Bourke, et al. 2012) extends the set $\mathbb{R}$ of real numbers into a superset $^\star\mathbb{R}$ of *hyperreals* (also called *nonstandard reals*) that includes infinite sets of infinitely large numbers and infinitely small numbers. Key properties of hyperreals, needed for the informal discussion of the Cup-and-Ball example, are the following:

There exist *infinitesimals*, defined as hyperreals that are smaller in absolute value than any real number: an infinitesimal $\partial \in {}^\star\mathbb{R}$ is such that $|\partial| < a$ for any positive $a \in \mathbb{R}$. For x, y two hyperreals, write $x \approx y$ if $x - y$ is an infinitesimal.

All relations, operators, and propositional formulas that are valid over $\mathbb{R}$ are also valid over $^\star\mathbb{R}$. For example, $^\star\mathbb{R}$ is a totally ordered set. The arithmetic operations $+$, $\times$, etc. can be lifted to $^\star\mathbb{R}$. We say that a hyperreal x is *finite* if there exists some standard finite positive real number a such that $|x| < a$.

For every finite hyperreal $x \in {}^\star\mathbb{R}$, there is a unique standard real number $\mathtt{st}(x) \in \mathbb{R}$ such that $\mathtt{st}(x) \approx x$, and $\mathtt{st}(x)$ is called the *standard part* (or *standardization*) of x. Standardizing more complex objects, such as functions or systems of equations, requires some care (see Theorem 1, Section 2.5).

Every real function lifts in a systematic way to a hyperreal function. This allows us to write $f(x)$ where f is a real function (regardless of its continuity properties) and x is a nonstandard number.

Continuity and derivatives. Let $t \mapsto x(t)$ be an $\mathbb{R}$-valued (standard) signal ($t \in \mathbb{R}$). Then:

x is continuous at instant $t \in \mathbb{R}$ if and only if, for any infinitesimal $\partial \in {}^\star\mathbb{R}$, one has $x(t + \partial) \approx x(t)$; (4)

x is differentiable at instant $t \in \mathbb{R}$ if and only if there exists $a \in \mathbb{R}$ such that, for any infinitesimal $\partial \in {}^\star\mathbb{R}$, $\frac{x(t+\partial) - x(t)}{\partial} \approx a$. In this case, $a = x'(t)$. (5)

We can then consider the time index set $\mathbb{T} \subseteq {}^\star\mathbb{R}$:

$$\mathbb{T} = 0, \partial, 2\partial, 3\partial, \cdots = \{n\partial \mid n \in {}^\star\mathbb{N}\} \qquad (6)$$

where ∂ is a positive infinitesimal, and $^\star\mathbb{N}$ denotes the set of *hyperintegers*, consisting of all integers augmented with additional infinite numbers called *nonstandard*. The important features of $\mathbb{T}$ are: (1) Any finite real time $t \in \mathbb{R}_+$, where $\mathbb{R}_+$ denotes the set of nonnegative real numbers, is infinitesimally close to some element of $\mathbb{T}$ (informally, $\mathbb{T}$ covers $\mathbb{R}_+$ and can be used to index continuous-time dynamics); and (2) $\mathbb{T}$ is "discrete": every instant $n\partial$ has a predecessor $(n-1)\partial$ (except for $n = 0$) and a successor $(n+1)\partial$.

Let x be a nonstandard signal indexed by $\mathbb{T}$. We define the *forward-* and *backward-shifted* signals $x^\bullet$ and $^\bullet x$ through

$$x^\bullet(n\partial) =_{\text{def}} x((n+1)\partial) \ \text{ and } \ {}^\bullet x((n+1)\partial) =_{\text{def}} x(n\partial) \ ,$$

implying that an initial value for $^\bullet x(0)$ must be provided. For f a function of the tuple X of signals, we set $(f(X))^\bullet =_{\text{def}} f(X^\bullet)$ where the forward shift $X \mapsto X^\bullet$ applies pointwise to all the components of the tuple. For example, $f^\bullet(x, y)(t) = f(x(t+\partial), y(t+\partial))$.

By (5), this allows us to represent, up to an infinitesimal, the derivative x' of a signal by its first-order explicit Euler approximation $\frac{1}{\partial}(x^{\bullet} - x)$. Solutions of multimode DAE systems may, however, be non-differentiable and even non-continuous at events of mode change. To give a meaning to x' at any instant, *we decide to **define** it everywhere as the nonstandard first-order Euler increment.*

Hence, we implicitly add to every system the following two equations, for each state variable x:

$$x' = \frac{x^{\bullet} - x}{\partial} \;\; ; \;\; x'' = \frac{x^{\bullet 2} - 2x^{\bullet} + x}{\partial^2} \; , \tag{7}$$

where

$$\begin{array}{rcl} x^{\bullet}(t) & =_{\mathrm{def}} & x(t+\partial)\,, \\ x^{\bullet 2}(t) & =_{\mathrm{def}} & x(t+2\partial) \text{ and, generally} \\ x^{\bullet n}(t) & =_{\mathrm{def}} & x(t+n\partial)\,. \end{array}$$

Equation (7) means that the derivatives x', y', x'', y'' are interpreted using the explicit first-order Euler scheme with an *infinitesimal time step* ∂. Note that (7) implies

$$x'' \quad = \quad \frac{x'^{\bullet} - x'}{\partial} \quad . \tag{8}$$

This yields the nonstandard expansion of the corrected model (3):

$$\left\{ \begin{array}{rll} & 0 = x'' + \lambda x & (e_1) \\ & 0 = y'' + \lambda y + g & (e_2) \\ & \gamma^{\bullet} = [s \leq 0]; \gamma(0) = \mathrm{F} & (k_0) \\ \texttt{if}\ \gamma\,\texttt{do} & 0 = L^2 - (x^2 + y^2) & (k_1) \\ \texttt{and} & 0 = \lambda + s & (k_2) \\ \texttt{if not}\ \gamma\,\texttt{do} & 0 = \lambda & (k_3) \\ \texttt{and} & 0 = (L^2 - (x^2 + y^2)) - s & (k_4) \end{array} \right. \tag{9}$$

This model is understood in the nonstandard setting, meaning that the derivatives are expanded using (7). Therefore, the leading variables in all modes are $\lambda, s, x^{\bullet 2}, y^{\bullet 2}$.

We are ready to concentrate on structural analysis and we will focus on the main difficulty with this Cup-and-Ball model, namely the mode change γ:F→T, when the rope gets straight. The reader is referred to the companion paper (Benveniste, Caillaud, and Malandain 2021) for omitted details.

2.3 Structural analysis of mode change γ:F→T

Due to equation (k_1), the mode $\gamma = \mathrm{T}$ (where the rope is straight) requires index reduction. We thus augment model (9) with the two latent equations shown in red:

$$\left\{ \begin{array}{rll} & 0 = x'' + \lambda x & (e_1) \\ & 0 = y'' + \lambda y + g & (e_2) \\ & \gamma^{\bullet} = [s \leq 0]; \gamma(0) = \mathrm{F} & (k_0) \\ \texttt{if}\ \gamma\,\texttt{do} & 0 = L^2 - (x^2 + y^2) & (k_1) \\ \texttt{and} & 0 = L^2 - (x^2 + y^2)^{\bullet} & (k_1^{\bullet}) \\ \texttt{and} & 0 = L^2 - (x^2 + y^2)^{\bullet 2} & (k_1^{\bullet 2}) \\ \texttt{and} & 0 = \lambda + s & (k_2) \\ \texttt{if not}\ \gamma\,\texttt{do} & 0 = \lambda & (k_3) \\ \texttt{and} & 0 = (L^2 - (x^2 + y^2)) - s & (k_4) \end{array} \right. \tag{10}$$

Note that the two latent equations $(k_1^{\bullet})$ and $(k_1^{\bullet 2})$ were obtained by *shifting* (k_1) *forward*, not by differentiating it as usually performed—the two, however, are equivalent from the structural analysis standpoint, because of equalities (7).

To perform structural analysis at the considered mode change, we first unfold model (10) at the successive instants

$${}^{\bullet 2}t \;=_{\mathrm{def}}\; t - 2\partial, \quad {}^{\bullet}t \;=_{\mathrm{def}}\; t - \partial, \quad \text{and } t\,,$$

where t denotes the current instant. In the following, equation (e_1) at the instant $t - 2\partial$ (respectively, $t - \partial$) will be denoted by $({}^{\bullet 2}e_1)$ (resp., $({}^{\bullet}e_1)$).

In this unfolding, the two equations (k_1) and $(k_1^{\bullet})$ are in structural conflict with selected equations from the previous two instants, shown in blue in the following subsystem, whose dependent variables are the leading variables at instants $t - 2\partial$ and $t - \partial$, namely $x, y, {}^{\bullet 2}\lambda; x^{\bullet}, y^{\bullet}, {}^{\bullet}\lambda$:

$$\left\{ \begin{array}{ll} 0 = \frac{x - 2\,{}^{\bullet}x + {}^{\bullet 2}x}{\partial^2} + {}^{\bullet 2}\lambda\; {}^{\bullet 2}x & ({}^{\bullet 2}e_1) \\ 0 = \frac{y - 2\,{}^{\bullet}y + {}^{\bullet 2}y}{\partial^2} + {}^{\bullet 2}\lambda\; {}^{\bullet 2}y + g & ({}^{\bullet 2}e_2) \\ 0 = \frac{x^{\bullet} - 2x + {}^{\bullet}x}{\partial^2} + {}^{\bullet}\lambda\; {}^{\bullet}x & ({}^{\bullet}e_1) \\ 0 = \frac{y^{\bullet} - 2y + {}^{\bullet}y}{\partial^2} + {}^{\bullet}\lambda\; {}^{\bullet}y + g & ({}^{\bullet}e_2) \\ 0 = L^2 - (x^2 + y^2) & (k_1) \\ 0 = L^2 - (x^2 + y^2)^{\bullet} & (k_1^{\bullet}) \end{array} \right.$$

This conflict can be detected from structural information only, using the Dulmage-Mendelsohn decomposition (Dulmage and Mendelsohn 1958). We propose to resolve this conflict by applying the following principle:

Principle 1 (Causality) *What was done at the previous instant cannot be undone at the current instant.*

Applying Principle 1 leads to erasing, in model (10), equations (k_1) and $(k_1^{\bullet})$ at the instant of mode change ${}^{\bullet}\gamma{=}\mathrm{F}, \gamma{=}\mathrm{T}$. This yields the following system:

$$\text{at} \left[\begin{array}{c} {}^{\bullet}\gamma{=}\mathrm{F} \\ \gamma{=}\mathrm{T} \end{array} \right] : \left\{ \begin{array}{ll} 0 = x'' + \lambda x & (e_1) \\ 0 = y'' + \lambda y + g & (e_2) \\ 0 = L^2 - (x^2 + y^2)^{\bullet 2} & (k_1^{\bullet 2}) \\ 0 = \lambda + s & (k_2) \end{array} \right. \tag{11}$$

It uniquely determines all the leading variables from the state variables x, y and $x^{\bullet}, y^{\bullet}$. In turn, equations (k_1) and $(k_1^{\bullet})$, which were erased from this model, are not satisfied.

At the next instant, i.e., when ${}^{\bullet 2}\gamma{=}\text{F}, {}^{\bullet}\gamma{=}\text{T}, \gamma{=}\text{T}$, the same argument is used. We thus erase, in model (10), the only equation (k_1) at the next instant. This yields the following system:

$$\text{at} \left[\begin{array}{c} {}^{\bullet 2}\gamma{=}\text{F} \\ {}^{\bullet}\gamma{=}\text{T} \\ \gamma{=}\text{T} \end{array}\right] : \left\{\begin{array}{ll} 0 = x'' + \lambda x & (e_1) \\ 0 = y'' + \lambda y + g & (e_2) \\ 0 = L^2 - (x^2+y^2)^{\bullet} & (k_1^{\bullet}) \\ 0 = L^2 - (x^2+y^2)^{\bullet 2} & (k_1^{\bullet 2}) \\ 0 = \lambda + s & (k_2) \end{array}\right. \tag{12}$$

Note that $(k_1^{\bullet})$ is a consistency equation that is satisfied by the state variables $x^{\bullet}, y^{\bullet}$. In turn, equation (k_1), which was erased from this model, is not satisfied. At subsequent instants, equation erasure is no longer needed: the process amounted to *delaying by a few nonstandard instants the satisfaction of some of the constraints in the new mode*, which actually took zero standard time. This completes the nonstandard structural analysis of the mode change $\gamma : \text{F} \rightarrow \text{T}$, i.e., when the rope gets straight.

2.4 Impulse analysis at mode change $\gamma : \text{F} \rightarrow \text{T}$

We now focus on identifying possible impulsive behaviors at this mode change. This is achieved by analyzing nonstandard systems (11) and (12) defining the values for restart. The intent is that the former will set the restart positions, whereas the latter will set the restart velocities.

Our impulse analysis not only identifies impulsive variables but also quantifies their order of magnitude, thanks to the following notion of impulse order:

Definition 1 (Impulse order and analysis)

1. *Given a nonstandard system of equations E defining the values for restart, say that a dependent variable x has* impulse order $\mathfrak{o} \in \mathbb{R}$ in *E, if the solution of system E is such that $x\partial^{-\mathfrak{o}}$ is provably a finite non-zero (standard) real number. Let $[\![x]\!]$ denote the impulse order of x. By convention, the constant* 0 *has impulse order* $-\infty$.

2. *Say that x is* impulsive *if* $[\![x]\!] > 0$.

3. *The* impulse analysis *of a system of equations S is the system of constraints satisfied by the impulse orders of the dependent variables of S.*

Impulse analysis relies on the following generic assumption, which expresses that DAE within long modes must be reinitialized with finite values for the state variables:

Assumption 1 *State variables are not impulsive; that is, for any state variable v, one has* $[\![v]\!] \leq 0$.

As an example, if, in the new mode, a variable x is differentiated up to order n, then its $(n-1)$-th derivative is a state variable and thus subject to Assumption 1. Consequently, its k-th order derivatives for $k = 0, \ldots, n-2$ are continuous at the considered mode change.

We are now ready to successively analyze Systems 11 and 12.

System (11): The state variables are x, y, x', y'. By Assumption 1, we get the following prior information, which expresses that velocities are not impulsive:

$$[\![x'^{\bullet} - x']\!] \leq 0 \quad ; \quad [\![y'^{\bullet} - y']\!] \leq 0 \ . \tag{13}$$

Conditions (13) imply that positions should be continuous. While performing our impulse analysis, we include equation (8) relating second derivatives and first derivatives. System (11) involves equation $(e_1) : x'' + \lambda x{=}0$, which, by using (8), rewrites

$$x'^{\bullet} - x' + \partial\lambda x = 0 \ . \tag{14}$$

By (13), equation (14) implies $[\![\lambda]\!] \leq 1$. Exploiting all equations of System (11) yields the following information

$$[\![\lambda]\!] = [\![s]\!] \leq 1 \ , \tag{15}$$

whereas other dependent variables have impulse order zero. System (12) is handled similarly, with the same conclusion. In Section 3, we mechanize the impulse analysis for an arbitrary restart system. Prior to doing this, we now explain how this impulse analysis can be exploited for generating effective code for restart.

2.5 Using impulse analysis in code generation

Code generation for restarts consists in standardizing nonstandard systems (11) and (12). See the introduction of Section 2.2 for the meaning of “standardization”; note, however, that standardizing systems of equations requires more care than standardizing numbers, due to impulsive behaviors and singularity issues that result.

We can exploit the impulse analysis through the following three different approaches. The method of Section 2.5.1 is mostly described for didactic purposes, as it requires the symbolic elimination of variables, which can be very costly or even impossible in nonlinear systems. In practice, the methods of Sections 2.5.2 and 2.5.3 shall be used; both of these sections briefly address this topic.

2.5.1 Eliminating impulsive variables

When this is practical, the simplest method from a conceptual point of view is to eliminate impulsive variables from the restart system, as they are of no use for restarting the new mode.

We still focus here on the standardization of the mode change $\gamma : \text{F} \rightarrow \text{T}$, i.e., when the rope gets straight. Our task is to standardize systems (11) and (12), by targeting discrete-time dynamics, for the two successive instants composing the restart phase. This will provide us with restart values for positions and velocities.

By (15), tensions λ and $\lambda^{\bullet}$ are both candidates to be impulsive, hence so are s and $s^{\bullet}$ by $(k_2, k_2^{\bullet})$. We eliminate the impulsive variables by ignoring $(k_2, k_2^{\bullet})$, combining (e_1) and (e_2) to eliminate λ, and $(e_1^{\bullet})$ and $(e_2^{\bullet})$ to eliminate $\lambda^{\bullet}$.

This yields:

$$\text{at}\left[\begin{array}{r}{}^{\bullet}\gamma{=}\mathrm{F}\\ \gamma{=}\mathrm{T}\end{array}\right] \;:\; \left\{\begin{array}{l}0=y''x+gx-x''y\\ 0=L^2{-}(x^2{+}y^2)^{\bullet 2}\end{array}\right. \tag{16}$$

$$\text{at}\left[\begin{array}{r}{}^{\bullet 2}\gamma{=}\mathrm{F}\\ {}^{\bullet}\gamma{=}\mathrm{T}\\ \gamma{=}\mathrm{T}\end{array}\right] \;:\; \left\{\begin{array}{l}0=y''x+gx-x''y\\ 0=L^2{-}(x^2{+}y^2)^{\bullet}\\ 0=L^2{-}(x^2{+}y^2)^{\bullet 2}\end{array}\right. \tag{17}$$

In System (16), we expand second derivatives using (7), whereas, in System (17), we expand them using (8). Consequently, System (16) has dependent variables $x^{\bullet 2}, y^{\bullet 2}$, whereas System (17) has dependent variables $x'^{\bullet}, y'^{\bullet}$. We are now ready to standardize the two systems.

System (16) to define restart positions: We expand second derivatives using (7):

$$\left\{\begin{array}{l}0=(y^{\bullet 2}-2y^{\bullet}+y)x-(x^{\bullet 2}-2x^{\bullet}+x)y+\partial^2 gx\\ 0=L^2{-}(x^2{+}y^2)^{\bullet 2}\end{array}\right. \tag{18}$$

Setting $\partial = 0$ in System (18) yields a structurally regular system, so that we can invoke the following result, proved in (Benveniste, Caillaud, and Malandain 2020):

Theorem 1 (standardizing systems of equations) *For* $\mathbf{H} : \mathbb{R}^{n+1} \to \mathbb{R}^n$ *a* $\mathscr{C}^1$ *(standard) function, consider the nonstandard system of equations* $\mathbf{H}(\partial, X) = 0$ *where* X *is a n-vector of variables. If system* $\mathbf{H}(0, X) = 0$ *is structurally nonsingular, then setting* $\partial = 0$ *in system* $\mathbf{H}(\partial, X) = 0$ *yields the correct standardization of it, meaning that the solution* $x_*(\partial)$ *of* $\mathbf{H}(\partial, X) = 0$ *standardizes as the solution* x_* *of* $\mathbf{H}(0, X) = 0$.

By this theorem, setting $\partial = 0$ in System (18) yields the correct standardization of it:

$$\left\{\begin{array}{l}0=(y^{\bullet 2}-2y^{\bullet}+y)x-(x^{\bullet 2}-2x^{\bullet}+x)y\\ 0=L^2{-}(x^2{+}y^2)^{\bullet 2}\end{array}\right.$$

Then, in the resulting system, we interpret x and $x^{\bullet}$ as the left-limit x^- of state variable x in previous mode, and $x^{\bullet 2}$ as the restart value x^+ for the new mode. This yields

$$\left\{\begin{array}{l}0=(y^+-y^-)x^- -(x^+-x^-)y^-\\ 0=L^2{-}(x^2{+}y^2)^+\end{array}\right. \tag{19}$$

which determines the restart values for positions. Note that the constraint that the rope is straight is satisfied. Furthermore, as $0 = L^2{-}(x^2{+}y^2)^-$ also holds (the rope is straight at the mode change), $x^+ = x^-, y^+ = y^-$ is the unique solution of (19): positions are continuous.

System (17) to define restart velocities: We expand second derivatives using (8):

$$\left\{\begin{array}{l}0=(y'^{\bullet}-y')x-(x'^{\bullet}-x')y+\partial.gx\\ 0=L^2{-}(x^2{+}y^2)^{\bullet}\\ 0=L^2{-}(x^2{+}y^2)^{\bullet 2}\end{array}\right. \tag{20}$$

By expanding $x^{\bullet 2} = x^{\bullet} + \partial x'^{\bullet}$, the right-hand side of the last equation rewrites

$$\begin{array}{rcl}L^2{-}(x^2{+}y^2)^{\bullet 2} = & & L^2{-}(x^2{+}y^2)^{\bullet}\\ & + & 2\partial(x^{\bullet}x'^{\bullet}+y^{\bullet}y'^{\bullet})\\ & + & \partial^2\big((x'^{\bullet})^2+(y'^{\bullet})^2\big)\\ = & & 0\ \text{(using (20))}\\ & + & 2\partial(x^{\bullet}x'^{\bullet}+y^{\bullet}y'^{\bullet})\\ & + & O(\partial^2)\end{array} \tag{21}$$

Using this expansion of $L^2{-}(x^2{+}y^2)^{\bullet 2}$, setting $\partial = 0$ in (20) yields

$$\left\{\begin{array}{l}0=(y'^{\bullet}-y')x-(x'^{\bullet}-x')y\\ 0=x^{\bullet}x'^{\bullet}+y^{\bullet}y'^{\bullet}\end{array}\right. \tag{22}$$

where the dependent variables are now $x'^{\bullet}, y'^{\bullet}$, whereas other variables are state variables whose values are determined by previous time steps. Note that System (22) is structurally regular, so that we can invoke Theorem 1, showing that System (22) is the correct standardization of System (20). We are now ready to get effective code for the restart. In System (22), we perform the following substitutions, where superscripts $^-$ and $^+$ denote left- and right-limits, and the continuity of positions is used:

$$x = x^- \ ;\ x^{\bullet} = x^+ \quad \text{and} \quad x' = x'^- \ ;\ x'^+ = x'^{\bullet} \tag{23}$$

and similarly for y. This finally yields

$$\left\{\begin{array}{l}0=(y'^+-y'^-)x^- -(x'^+-x'^-)y^-\\ 0=x^+x'^+ + y^+y'^+\end{array}\right. \tag{24}$$

System (24) determines x'^+ and y'^+, which are the velocities for restart. The second equation guarantees that the velocity will be tangent to the constraint. With (19) and (24), we determine the restart conditions for positions and velocities. Invariants from the physics are satisfied.

This is a satisfactory solution when the elimination of impulsive variables is practical. In our example, they entered linearly in the restart system, so that elimination was straightforward. When this is not the case, elimination becomes costly or even impossible. Moreover, generalizing and mechanizing this elimination process appears to be a very difficult task. We thus need to look for alternatives for computing the velocities for restart.

2.5.2 Rescaling impulsive variables

Focus again on System (12). Impulse analysis told us that λ, s both have impulse order ≤ 1. We thus rescale them accordingly:

$$\widehat{\lambda} =_{\text{def}} \partial^1 \times \lambda \quad \text{and} \quad \widehat{s} =_{\text{def}} \partial^1 \times s \tag{25}$$

Using this rescaling together with expansion (8), System (12) rewrites

$$\left\{\begin{array}{ll} 0 = x'^{\bullet} - x' + \widehat{\lambda}x & (e_1) \\ 0 = y'^{\bullet} - y' + \widehat{\lambda}y + \partial g & (e_2) \\ 0 = L^2 - (x^2+y^2)^{\bullet} & (k_1^{\bullet}) \\ 0 = L^2 - (x^2+y^2)^{\bullet 2} & (k_1^{\bullet 2}) \\ 0 = \widehat{\lambda} + \widehat{s} & (k_2) \end{array}\right. \tag{26}$$

In System (26), $(k_1^{\bullet})$ is a consistency equation satisfied as a result of performing System (11) at the previous instant. We can also discard equation (k_2), which only serves to determine the auxiliary variable s. Thus, we are left with the sub-system collecting equations $(e_1), (e_2), (k_1^{\bullet 2})$. We can again expand the right-hand side of $(k_1^{\bullet 2})$ by using (21). In the resulting system, we can safely set $\partial \leftarrow 0$ since it yields the following structurally regular system:

$$\left\{\begin{array}{ll} 0 = x'^{+} - x'^{-} + \widehat{\lambda}x^{-} & (e_1) \\ 0 = y'^{+} - y'^{-} + \widehat{\lambda}y^{-} & (e_2) \\ 0 = 0 = x^{+}x'^{+} + y^{+}y'^{+} & (k_1^{\bullet 2}) \end{array}\right. \tag{27}$$

System (27) determines $x'^{+} = x'^{\bullet}, y'^{+} = x'^{\bullet}$, and the rescaled impulsive tension $\widehat{\lambda}$, as functions of state variables x', y', x, y, which were identified with the left-limits of velocities and positions at previous mode. Note that eliminating the rescaled tension $\widehat{\lambda}$ from System (27) yields System (24).

Rescaling impulsive variables is simpler than eliminating them. This method is also promising in terms of designing and implementing algorithms for its mechanization, as the computation of the impulse orders amounts to finding a minimal solution to a system of linear unilateral constraints. Unfortunately, it does not work in full generality since impulse orders can be infinite, as the following example shows:

$$x = \exp(y/\partial),$$

where y is known to have impulse order zero. Indeed, the impulse order of $(y/\partial)^n$ is n. Since the exponential expands as a power series of infinite support, we deduce that the impulse order of $\exp(y/\partial)$ is the maximum of all impulse orders of $(y/\partial)^n$, hence it is infinite. Thus, impulsive variable x cannot be rescaled.

The last method addresses such cases, at the price of a possibly poor numerical conditioning.

2.5.3 Bruteforce solving of the restart system

When none of the above methods apply, it is still possible to solve system (26) with $\partial = \delta$ (a small positive time step) for the original variables λ and s, without rescaling them.

Then, it is proved in (Benveniste, Caillaud, and Malandain 2020), see also (Benveniste, Caillaud, and Malandain 2021) that *solving these systems for their dependent variables and then discarding the values found for the impulsive variables yields a converging approximation for the states and velocities at restart.* Moreover, first numerical experiments on toy examples showed no issue as long as the time step δ was kept reasonably high. Of course, without rescaling, the numerical conditioning is likely to be less favorable, so that rescaling is recommended when impulse orders are finite. Works are in progress for the implementation of this method, coupled with the rescaling of impulsive variables of finite order.

2.6 Handling transient modes: elastic impact

Our reasoning so far produces a behavior in which the two modes (free motion and straight rope) gently alternate; the system always stays in one mode for some positive period of time before switching to the other mode.

This indeed amounts to assuming that the impact is totally inelastic at mode change, an assumption that was not explicit at all in (9). So, what happened? In fact, *the straight rope mode was implicitly assumed to last for at least three nonstandard successive instants, since we allowed ourselves to shift* (k_1) *forward twice.*

Now, let us instead assume elastic impact, represented by the cascade of mode changes $\gamma : \mathrm{F} \to \mathrm{T} \to \mathrm{F}$, reflecting that the straight rope mode is *transient* (it is left immediately after being reached).

We address transient modes in (Benveniste, Caillaud, and Malandain 2020; Benveniste, Caillaud, and Malandain 2021). We show that a structural analysis for elastic impact can still be proposed, by suitably adapting the notion of *differentiation array* proposed by Campbell and Gear (1995). The so obtained structural analysis proves that our original model (1) for the Cup-and-Ball is underspecified at mode change $\gamma : \mathrm{F} \to \mathrm{T}$, when the rope gets straight. This underdetermination implies that the model is ill-defined, as it admits an infinite number of solutions. Completing it by adding an impact law, which makes sense from a physicist's point of view, is also appropriate from the point of view of our structural analysis.

One possible choice is to complete the model with an elastic impact law. This indeed corrects the restart system at $\gamma = \mathrm{T}$ in the cascade of mode changes $\gamma : \mathrm{F} \to \mathrm{T} \to \mathrm{F}$, yielding

$$\left\{\begin{array}{ll} 0 = x'' + \lambda x & (e_1) \\ 0 = y'' + \lambda y + g & (e_2) \\ 0 = y'^{\bullet} + (1-\alpha)y' & (\tau_1) \\ 0 = L^2 - (x^2+y^2) & (k_1) \\ 0 = \lambda + s & (k_2) \end{array}\right. \tag{28}$$

where $0 < \alpha < 1$ is a damping factor. We proceed again with the structural analysis. Variables x, y are the states, so that their values are set by the previous instants. Current equation (k_1) creates a conflict with the past. Hence, we discard it from System (28), which leaves us with the following system:

$$\left\{\begin{array}{ll} 0 = x'' + \lambda x & (e_1) \\ 0 = y'' + \lambda y + g & (e_2) \\ 0 = y'^{\bullet} + (1-\alpha)y' & (\tau_1) \\ 0 = \lambda + s & (k_2) \end{array}\right. \tag{29}$$

Model (29) is structurally nonsingular, recalling that y'' and $y'^{\bullet}$ can be interchanged for the structural analysis. This refined model is therefore accepted by the structural analysis.

The impulse analysis proceeds as for the previous case of inelastic impact and effective code for restart can be generated.

Note that any impact law could be used instead of the one added in System (28), as long as it ensures uniqueness of the solution for a fixed state before the impact.

In (Benveniste, Caillaud, and Malandain 2021), we also discuss the consequences, for modeling languages such as Modelica, of the need for stating as a side specification whether a mode is transient or not.

3 General Impulse Analysis

In this section, we explain how the reasoning used for the Cup-and-Ball example can be mechanized as a compilation stage following multimode structural analysis. Prior to developing this, we provide a simplified overview of said multimode structural analysis.

3.1 Overview of multimode structural analysis

We consider multimode DAE systems possessing *long modes* (having DAE-based dynamics for a positive duration) alternating with finite cascades of *transient modes* (having a zero duration, such as the straight rope mode in the Cup-and-Ball model with elastic impact).

We assume that the information regarding the type of a mode (long vs. transient) is known by the compiler—the two different Modelica primitives `if` and `when` should be used to declare long and transient modes, respectively.

In addition, we require that the current mode is defined by the left-limits of some predicates, see the reasoning leading to the corrected model (9) for the Cup-and-Ball.

For such models, the structural analysis proceeds according to the following steps:

1. The multimode model is mapped to its nonstandard expansion by using a first-order explicit Euler expansion for derivatives, with infinitesimal time step ∂, and mapping left-limits to values at the previous instant. In particular, the mode at each nonstandard instant is known at the end of the previous instant.

2. The structural analysis for each specific mode is performed, depending on its long/transient type:

   - If the mode is long, then classical structural analysis applies: by, e.g., using Pryce's Σ-method (Pryce 2001), latent equations are added for the DAE system associated to each long mode;
   - Alternatively, if the mode is transient, a structural analysis of the *difference array* associated to the considered cascade of transient modes is performed.

3. Having done this, given the mode at the current instant:

   - If no mode change occurs, then the (classical) mode-specific structural analysis applies;
   - Otherwise, the conflict that may possibly exist between consistency equations of the current mode and leading equations of the previous mode is analyzed, using the Dulmage-Mendelsohn decomposition; conflicting subsystems are identified and the equations from the current instant that cause conflicts are erased.

Implementing the multimode structural analysis in the above described form would be very inefficient. Fortunately, Caillaud, Malandain, and Thibault (2020) proposed a very efficient algorithm for handling all the long modes simultaneously without enumerating them, and extended the Σ-method in this “all-modes-at-once” framework. A similar extension of the Dulmage-Mendelsohn decomposition is being implemented.

3.2 General Rules of Impulse Analysis

3.2.1 Problem setting

Restart systems of equations, as resulting from the structural analysis at mode changes, are nonstandard systems of equations of the following generic form:

$$\text{expand } X' \text{ as } \frac{X^{\bullet}-X}{\partial} \text{ in } 0 = \mathbf{H}(X', X^{\bullet}, V, X) \qquad (30)$$

where V collects the algebraic variables, X collects the state variables, and $\frac{X^{\bullet}-X}{\partial}$ is the nonstandard semantics of X'. $\mathbf{H}(\cdot,\cdot,\cdot,\cdot)$, seen as a vector function in its dotted arguments, is by itself standard, since the equations of system $0 = \mathbf{H}$ are obtained by shifting or differentiating equations specified by the user. The reason for (30) being nonstandard is indeed twofold:

1. Since $X^{\bullet}$ is involved, the infinitesimal ∂ occurs in time; and

2. Since X' is involved, the infinitesimal ∂ occurs both in time and *space,* due to the expansion $X' \leftarrow \frac{X^{\bullet}-X}{\partial}$.

The occurrence of ∂ in time is not an issue: shifted state variables will correspond to restart values for states, whereas non-shifted ones correspond to values prior to the change. In contrast, the occurrence of ∂ in space is the root cause of possible impulsive behaviors. Identifying them is the subject of impulse analysis.

3.2.2 The rules of impulse analysis

We now develop the *impulse analysis* introduced in Definition 1. This analysis is useful as a postprocessing of structural analysis, prior to generating effective code for restarts. Note that Assumption 1 is still enforced in what follows.

Figures 2 and 3 display the rules defining the translation of a system of equations of the form (30) into its impulse analysis, for the restricted class where only rational expressions are involved.

Figure 2 describes the syntax of a mini-language specifying such systems of equations. The left column of Figure 3 gives the rules for mapping expressions to their corresponding impulse orders. The reason for the inequality in (R6) is that in the sum $e_1{+}e_2$, the dominant terms in the expansions of e_i as series over ∂ may cancel each other. For an example of this, see equation (e_2) in System (12): rewriting this equation as $-g = y'' + \lambda y$, we see a case of strict inequality for (R6) since gravity g has order zero, whereas it is equal to the difference of two terms of order one.

We will use Rule (R6) in the following way, thereby reinforcing it. Consider an equation

$$e : z = x + y \quad .$$

We can rewrite e in the following equivalent ways: $0 = x + y - z, x = z - y$, or $y = z - x$. To each of them we apply the max rule. This yields the following system of constraints, called the *impulse analysis of equation e*:

$$\left\{ \begin{array}{l} [\![z]\!] \leq \max\{[\![x]\!], [\![y]\!]\} \\ [\![0]\!] \leq \max\{[\![x]\!], [\![y]\!], [\![z]\!]\} \\ [\![x]\!] \leq \max\{[\![z]\!], [\![y]\!]\} \\ [\![y]\!] \leq \max\{[\![x]\!], [\![z]\!]\} \end{array} \right. \tag{31}$$

Note that the constraint $[\![0]\!] \leq \ldots$ is vacuously satisfied since $[\![0]\!] = -\infty$. Then, among the three nontrivial inequalities of (31), at least two of them must be saturated. We will use impulse analysis (31) for handling sums of terms. This reinforcement of the max rule is formalized by Rule (R8) of Figure 3, which mechanizes the association, to any equation, of its different rewritings.

Using the rules of Figures 2 and 3 in the numerical expressions, we map any system of rational equations of the form (30) into a system of constraints over impulse orders.

To cover functions beyond polynomials, we need to extend $\mathbb{R} \cup \{-\infty\}$ with $+\infty$. In this extension, we take the convention that $-\infty + \infty = -\infty$, justified by both Rules (R1,R5) and the equality $0{\times}x = 0$ for any nonstandard x. For functions $f(x) = \sum_{k=0}^{\infty} a_k x^k$ that can be represented as absolutely converging power series, we then get

$$[\![f(x)]\!] = [\![\sum_{k=0}^{\infty} a_k x^k]\!] = [\![x]\!].\sup(A) \quad , \tag{32}$$

where $A{=}\{k \mid a_k{\neq}0\}$ is the support of the series and $\sup(A)$ is the supremum of set A. In particular, if $[\![x]\!] > 0$ and if the support of the series is infinite, we get $[\![f(x)]\!] = +\infty$.

3.2.3 Particularizing the impulse analysis to systems of equations for restarts

So far, Rules (R1)–(R8) of the impulse analysis apply to any system of nonstandard equations. Here we particularize the impulse analysis to systems of equations of the form (30), where the only reason for ∂ to occur is the expansion of derivatives using the Euler scheme:

$$0 = \mathbf{H}\left(\frac{X^\bullet - X}{\partial}, X^\bullet, V, X\right)$$

The dependent variables are $X^\bullet, V$. It will be convenient to introduce the auxiliary variables

$$U =_{\text{def}} X^\bullet - X \, ,$$

so that the systems we consider take the following form, where $X^\bullet, V, U$ are the dependent variables:

$$\left\{ \begin{array}{lcl} 0 & = & \mathbf{H}\left(\frac{U}{\partial}, X^\bullet, V, X\right) \\ U & = & X^\bullet - X \end{array} \right. \tag{33}$$

The following condition for System (33) can be assumed, based on physical considerations (restart values for an ODE or a DAE cannot be impulsive):

Assumption 2 *Since X is a state, both X (a known value) and $X^\bullet$ must be finite.*

First, the impulse orders $[\![X]\!]$ are all known, from previous nonstandard instants. Next, from Assumption 2 we deduce the inequalities:

$$[\![X^\bullet]\!] \leq 0 \quad \text{and} \quad [\![U]\!] \leq 0 \, . \tag{34}$$

The impulse orders $[\![V]\!]$ are a priori unknown. We have, however, more prior information, thanks to the structural analysis. From the structural analysis at the considered mode change, we know which consistency equations of the new mode were conflicting with the dynamics of previous mode. Formally, call $G{=}0$ the subsystem collecting all the equations that were erased while solving this conflict—for the Cup-and-Ball model (10), at the instant of mode change ${}^\bullet\gamma{=}\text{F}, \gamma{=}\text{T}$, G collects the bodies of the two violated consistency constraints (k_1) and $(k_1^\bullet)$.

As a result, $G{=}0$ no longer holds at the considered mode change, and thus, G defines a tuple R of variables (one per entry of G) called *residuals*, by setting

$$R = G, \tag{35}$$

which are all finite and nonzero. An example of residual in the Cup-and-Ball is $r = L - (x^2 + y^2)$, which is both finite and nonzero at mode change ${}^\bullet\gamma{=}\text{F}, \gamma{=}\text{T}$. The residuals are found by the structural analysis.

Finally, the system of equations that we need to solve collects all the above items, namely:

$$\left\{ \begin{array}{lcl} 0 & = & \mathbf{H}\left(\frac{U}{\partial}, X^\bullet, V, X\right) \\ U & = & X^\bullet - X \\ R & = & G \end{array} \right. \tag{36}$$

$$\begin{array}{rcl} e & ::= & 0 \mid c \mid \partial \mid x \mid e^c \mid e+e \mid e\times e \\ E & ::= & e=e \mid E \ \texttt{and}\ E \end{array}$$

Figure 2. Syntax: E is a system of one or several equations $e=e$. An expression e is 0, a nonzero (standard) real constant c, the infinitesimal ∂, a variable x, the monomial e^c, a sum, or a product.

$$\begin{array}{lrcl} \text{(R1)} & [\![0]\!] & = & -\infty \\ \text{(R2)} & [\![c]\!] & = & 0 \\ \text{(R3)} & [\![\partial]\!] & = & -1 \\ \text{(R4)} & [\![e^c]\!] & = & c[\![e]\!] \\ \text{(R5)} & [\![e_1\times e_2]\!] & = & [\![e_1]\!]+[\![e_2]\!] \\ \text{(R6)} & [\![e_1+e_2]\!] & \leq & \max\{[\![e_1]\!],[\![e_2]\!]\} \end{array}$$

$$\frac{E\vdash e=e'}{[\![E]\!]\vdash[\![e]\!]=[\![e']\!]} \qquad \text{(R7)}$$

$$\frac{\left.\begin{array}{l} E\vdash x=y+e \ \text{ or} \\ E\vdash 0=y-x+e \end{array}\right\} \text{and } E\nvdash y=x-e}{E\vdash E \ \texttt{and}\ y=x-e} \qquad \text{(R8)}$$

Figure 3. Rules: The left column displays the impulse order of the primitive expressions. Rule (R7) indicates that $[\![e]\!]=[\![e']\!]$ is an equation of the impulse analysis $[\![E]\!]$ if $e=e'$ is an equation of E; rule (R8) indicates that, if E involves the equation $x=y+e$ but not the equation $y=x-e$, then we augment E with the latter, i.e., we saturate E with the rule $x=y+e \implies y=x-e$.

with dependent variables $X^\bullet, V, U, R$, and the following prior information on impulse orders is known:

$$[\![\frac{1}{\partial}]\!]=1\ ;\ [\![X^\bullet]\!]\leq 0\ ;\ [\![U]\!]\leq 0\ ;\ [\![R]\!]=0\,. \qquad (37)$$

System (36) is then mapped to its impulse analysis by using Rules (R1–R8) of Figures 2 and 3. A suitable constraint solver is then used to solve the resulting set of constraints on impulse orders, by using side information (37). The choice of an appropriate constraint solver remains to be done.

4 Conclusion

The correct handling of truly multimode Modelica models (in which index and structure may vary with the mode) requires significant add-ons to the existing structural analyses. The companion paper (Benveniste, Caillaud, and Malandain 2021) introduces, by means of two small but representative examples, a truly multimode structural analysis that applies both in modes and at mode changes. One important difficulty is the correct handling of impulsive behaviors for some variables.

In this paper, we introduced the *impulse analysis* of multimode DAE systems, a complement to multimode structural analysis for Modelica models. Impulse analysis is performed at compile time, prior to generating simulation code. It allows to identify impulsive variables, along with the mode changes at which impulsive behavior occurs. When impulsive behaviors occur in a model, then the conditions for restart at the impulsive mode change are generally known implicitly, not explicitly. Generating simulation code for restarts can thus be problematic. Using our approach based on impulse analysis, impulsive variables can be properly rescaled, so that correct explicit code for restarts can be generated.

In this paper, we did not consider the computational cost of performing true multimode structural analysis at compile time: unfortunately, the number of modes tends to be roughly exponential in the size of the model, and the *a priori* number of mode changes is at least proportional to the square of the number of modes. This is a limitation of a model representation in which one characterizes the subset of equations and variables active in any given mode.

A possible way of alleviating this issue is by shifting to a dual representation, that provides predicates characterizing the set of modes in which each equation and each variable is active. In practice, not only does this approach lead to a much more compact representation, but it also allows for the design of efficient structural analysis methods for multimode DAE systems, working in an 'all-modes-at-once' fashion. Such a method was implemented in the IsamDAE tool, and first results are reported in (Caillaud, Malandain, and Thibault 2020). The examples coming with this tool already include thermodynamical, electrical and pneumatic models. Although only the structural analysis of long modes is currently performed, the implementation of the structural analysis of mode changes is in progress.

Acknowledgements

The authors are indebted to several colleagues and friends for valuable discussions: Hilding Elmqvist and Martin Otter, John Pryce, and Vincent Acary. Khalil Ghorbal contributed to the first version of this approach.

This work was supported by the FUI ModeliScale DOS0066450/00 French national grant (2018-2021) and the Inria IPL ModeliScale large scale initiative (2017-2021, `https://team.inria.fr/modeliscale/`).

References

Acary, Vincent and Bernard Brogliato (2008). *Numerical Methods for Nonsmooth Dynamical Systems. Applications in Mechanics and Electronics*. Vol. 35. Lecture Notes in Applied and Computational Mechanics. Springer-Verlag. ISBN: 978-90-481-9680-7.

Benveniste, Albert, Timothy Bourke, et al. (2012). “Nonstandard semantics of hybrid systems modelers”. In: *J. Comput. Syst. Sci.* 78.3, pp. 877–910. DOI: 10.1016/j.jcss.2011.08.009.

Benveniste, Albert, Benoit Caillaud, Hilding Elmqvist, et al. (2019). “Multi-Mode DAE Models - Challenges, Theory and Implementation”. In: *Computing and Software Science - State of the Art and Perspectives*. Ed. by Bernhard Steffen and Gerhard J. Woeginger. Vol. 10000. Lecture Notes in Computer Science. Springer, pp. 283–310. ISBN: 978-3-319-91907-2. DOI: 10.1007/978-3-319-91908-9_16.

Benveniste, Albert, Benoit Caillaud, and Mathias Malandain (2020). “The mathematical foundations of physical systems modeling languages”. In: *Annual Reviews in Control* 50, pp. 72–118. ISSN: 1367-5788. DOI: 10.1016/j.arcontrol.2020.08.001.

Benveniste, Albert, Benoit Caillaud, and Mathias Malandain (2021-09). “Handling Multimode Models and Mode Changes in Modelica”. In: *Proceedings of the 14th International Modelica Conference*. Linköping University Electronic Press.

Caillaud, Benoit, Mathias Malandain, and Joan Thibault (2020-04). “Implicit Structural Analysis of Multimode DAE Systems”. In: *23rd ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2020)*. Sydney, Australia. DOI: 10.1145/3365365.3382201.

Campbell, Stephen L. and C. William Gear (1995). “The index of general nonlinear DAEs”. In: *Numer. Math.* 72, pp. 173–196.

Casella, Francesco (2015-09). “Simulation of Large-Scale Models in Modelica: State of the Art and Future Perspectives”. In: *Proc. of the Int. Modelica Conference*. Ed. by Hilding Elmqvist and Peter Fritzson. Modelica Association. Versailles, France.

Cellier, François and Ernesto Kofman (2006). *Continuous System Simulation*. Springer. ISBN: 9780387261027.

Cutland, Nigel (1988). *Nonstandard analysis and its applications*. Cambridge Univ. Press.

Dulmage, Andrew L. and Nathan S. Mendelsohn (1958). “Coverings of Bipartite Graphs”. In: *Canadian Journal of Mathematics* 10, pp. 517–534. DOI: 10.4153/CJM-1958-052-0.

Elmqvist, Hilding et al. (2012-09). “State Machines in Modelica”. In: *Proc. of the Int. Modelica Conference*. Ed. by Martin Otter and Dirk Zimmer. Modelica Association. Munich, Germany, pp. 37–46.

Liberzon, Daniel and Stephan Trenn (2012). “Switched nonlinear differential algebraic equations: Solution theory, Lyapunov functions, and stability”. In: *Automatica* 48.5, pp. 954–963. DOI: 10.1016/j.automatica.2012.02.041.

Lindstrøm, Tom (1988). “An Invitation to Nonstandard Analysis”. In: *Nonstandard Analysis and its Applications*. Ed. by N.J. Cutland. Cambridge Univ. Press, pp. 1–105.

Mattsson, Sven Erik, Martin Otter, and Hilding Elmqvist (1999). “Modelica Hybrid Modeling and Efficient Simulation”. In: *38th IEEE Conference on Decision and Control*. Ed. by IEEE, pp. 3502–3507.

Mattsson, Sven Erik and Gustaf Soderlind (1993). “Index Reduction in Differential-Algebraic Equations Using Dummy Derivatives”. In: *SIAM Journal on Scientific Computing* 14.3, pp. 677–692. DOI: 10.1137/0914043.

Pantelides, Constantinos C. (1988). “The consistent initialization of differential-algebraic systems”. In: *SIAM J. Sci. Stat. Comput.* 9.2, pp. 213–231.

Pryce, John D. (2001). “A simple structural analysis method for DAEs”. In: *BIT* 41.2, pp. 364–394.

Robinson, Abraham (1996). *Nonstandard Analysis*. Princeton Landmarks in Mathematics. ISBN: 0-691-04490-2.

Trenn, Stephan (2009a). “Distributional Differential Algebraic Equations”. PhD thesis. Technischen Universität Ilmenau.

Trenn, Stephan (2009b). “Regularity of distributional differential algebraic equations”. In: *MCSS* 21.3, pp. 229–264. DOI: 10.1007/s00498-009-0045-4.

A Modular Model of Reversible Heat Pumps and Chillers for System Applications

Fabian Wüllhorst[1] David Jansen[1] Philipp Mehrfeld[1] Dirk Müller[1]

[1]Institute for Energy Efficient Buildings and Indoor Climate, E.ON Energy Research Center, RWTH Aachen University, Germany, fabian.wuellhorst@eonerc.rwth-aachen.de

Abstract

Vapour compression machines such as heat pumps and chillers are vital for achieving climate goals. Efficiency of both depend mostly on system integration. In order to simulate coupled energy systems, fast and stable simulation models are required. Hence, we implement an open-source model for reversible vapour compression machines. The black-box based refrigeration cycle is replaceable, additional inertia and losses are optional. Furthermore, we model relevant safety controls of vapour compression machines. To show validity of the presented approach, we first calibrate two different black-box models onto measured heat pump data. The table-based model fits both measured temperature and power with minimal calibration effort. Second, we show influences of different model options onto coupled building performance simulations. Computation time increases up to 50 % when enabling all model options. Simultaneously, seasonal efficiency decreases by up to 23 % when modeling all safety controls. *Keywords: Heat Pump, Chiller, Modular Model*

1 Introduction

Heating and cooling account for half of Europe's secondary energy demand (Heat Roadmap Europe 2017). While cooling is already mainly achieved using electrically driven chillers (e.g. air conditioners), heat pumps (HPs) are set to replace gas or oil fired boilers (IEA 2020). Thus, improving efficiency of both is a major goal of current research.

As both chiller and heat pump are based on the same vapour compression cycles, designation mainly depends on which side contains the usable heat flow. Using a four-way reversing valve, a heat pump may act as a chiller and vice versa. Hence, in the following we refer to both as vapour compression machines (VCM).

To increase efficiency of VCMs, improvements on component level and system integration level are possible. Generally, the efficiency depends on the applied temperature levels. These temperature levels on the other hand depend mainly on the system integration. Thus, we focus on the integration of VCM into an energy system.

To thoroughly assess efficiency on system level, seasonal coefficients of performance (SCOPs) are required (Huchtemann and Dirk Müller 2012). While Hardware-in-the-Loop experiments as in Mehrfeld, Nürenberg, Knorr, et al. (2020) or field tests are cost intensive, model-based simulation analysis offers a viable solution to not only assess the SCOP but also to find promising control and system designs that optimize efficiency.

A model is always created based on the aim of the simulation analysis (VDI 3633:2014-12 2014). Consequently, numerous approaches for the modeling of VCMs exist in literature. The following section will give a brief overview.

2 Related work

The main distinction in models is between two types: Modeling the refrigeration cycle with empirical data (black-box) and using physical-based equation (gray/white-box) (Jin 2002). Black-box models are usually robust and require less computation time compared to the latter. In most cases, empirical data is based on steady-state operating points. Outside the given performance maps, which are often small, extrapolations usually yield non-physical results (Cimmino and Wetter 2017). However, for certain investigation aims, partly higher accuracies are achieved compared to gray-box models (Carbonell Sánchez et al. 2012). Afjei and Dott (2011) assign investigation targets to the different model types. Gray-box models are, due to the high level of detail and the high flexibility, suitable for the investigation of new heat pump concepts. For dynamic simulations on the other hand, black-box models with dynamic effects are useful (Afjei and Dott 2011).

2.1 Gray-box models

Cimmino and Wetter (2017) introduce two gray-box models, `ScrollWaterToWater` and `ReciprocatingWaterToWater`. By assuming a simplified refrigeration circuit, the modeling of the expansion valve is not required. Both models are non-reversible. As refrigerant, R410A is used. HP internal safety control is taken into account using temperature protection. By calibrating the model with manufacturer data, the parameters are defined and made accessible via records (Cimmino and Wetter 2017). Dechesne et al. (2017) model an inverter-controlled air/water HP with consideration of the refrigeration circuit for the refrigerant R410A. The possible icing of the evaporator

DOI
10.3384/ecp21181561

is taken into account through various assumptions. However, a validation of these assumptions is not performed (Dechesne et al. 2017). The model of a heat pump with water as refrigerant is implemented by Chamoun, Rulliere, Haberschill, and Berail (2012) for industrial applications. In another contribution, they introduce a screw compressor model for HP applications (Chamoun, Rulliere, Haberschill, and Peureux 2013).

2.2 Black-box models

In the following, two black-box models of the IBPSA library will be described (International Building Performance Simulation Association 2018). In each, the two variables electricity consumption P_{el} and used heat flow $\dot{Q}_{\text{Con}}$ are determined. Assuming a steady-state, adiabatic process, $\dot{Q}_{\text{Eva}}$ is then calculated:

$$0 = P_{\text{el}} + \dot{Q}_{\text{Eva}} - \dot{Q}_{\text{Con}} \tag{1}$$

The models Carnot_TCon and Carnot_y are based on the same principle, only the input variable changes. Carnot_TCon sets the output temperature of the condenser $T_{\text{Con,out}}$ via an ideal heater. Using the output $\dot{Q}_{\text{Con}}$ and the COP, P_{el} and $\dot{Q}_{\text{Eva}}$ are calculated. Carnot_y determines P_{el} by the product of $P_{\text{el,Nom}}$ and the input y which is the part load set point between zero and one. For both models, the COP is obtained using the Carnot efficiency at nominal and current conditions. These model approaches are valid for both chillers and heat pumps. For chillers, condenser and evaporator are switched (International Building Performance Simulation Association 2018).

For heat pumps, the AixLib library introduced an approach obtaining P_{el} and $\dot{Q}_{\text{Con}}$ through manufacturer data based on EN 14511 (EN 14511-1:2018-03 2018; D. Müller et al. 2016). Based on the same approach, the IDEAS library contains a heat pump model as well (Jorissen et al. 2018).

Besides these open-source models, researchers have presented their approach in various contributions. De Coninck et al. (2010) calculates P_{el} and $\dot{Q}_{\text{Con}}$ based on the ambient temperature T_{amb}, the condenser temperature T_{Con}, and manufacturer data. Wystrcil and Kalz (2012) use field test data to determine a four coefficient polynomial of the COP as a function of the temperatures $T_{\text{Eva,in}}$ and $T_{\text{Con,out}}$.

Until now, we focused our review on Modelica models. However, the modeling approaches in other simulation tools mostly do not differ. The assumption in Equation 1 is state of research for black-box approaches. Examples are given in (Afjei and Wetter 1997; EN 12900:2013-10 2013).

This review highlights the gap in the current state of the art, which can be summarized in two points.

1. Each contribution develops a new model. However, the black-box approaches only differ in how P_{el} and $\dot{Q}_{\text{Use}}$ are calculated. To spend less time in model development and more time in simulation analysis, a uniform and modular approach is necessary.
2. Besides Cimmino and Wetter (2017), VCM internal safety controls are disregarded. However, such controls strongly influence the behaviour of the machine in the energy system and thus, it's SCOP (Mehrfeld, Nürenberg, Knorr, et al. 2020).

Following the identified gaps, we present a new modular model for reversible heat pumps and chillers. The vapour compression cycle is modeled as a black-box. Contrary, the interactions with the energy systems are gray-box based. The model is presented in the following section.

3 Modular modeling approach

The following section presents our modeling approach. The overall model is depicted in Figure 1. To ensure reuseability of the approach, we implement all presented models in the open-source library AixLib (D. Müller et al. 2016). We aggregate all relevant inputs and outputs using a pre-defined expandable bus connector. All bus variables are listed in Table 1. In general, Set indicates a control variable and Mea a physical variable. As core of each VCM, we start with the black-box approach of the vapour compression cycle.

Table 1. Pre-defined variables of the VCM bus.

Variable	Explanation
nSet	Relative compressor frequency
modeSet	Set to false to reverse the device
onOffMea	Boolean indicating if device is on
PelMea	Measured electrical power
CoPMea	Measured COP
iceFacMea	Icing factor f_{ice}, see subsection 3.1
TOdaMea	Outdoor air temperature
TEvaInMea	Evaporator inlet temperature
TEvaOutMea	Evaporator outlet temperature
TEvaAmbMea	Ambient temperature at evaporator
TConInMea	Condenser inlet temperature
TConOutMea	Condenser outlet temperature
TConAmbMea	Ambient temperature at condenser

3.1 Vapour compression cycle

In order to realize a reversible and modular VCM model, we introduce a partial model named PartialPerformanceData. Inputs to the model are all bus variables listed in Table 1. Outputs are the three main energy flows, $\dot{Q}_{\text{Con,out}}$, $\dot{Q}_{\text{Eva,out}}$ and $P_{\text{el,out}}$. Both HP and chiller models may extend this model and connect these inputs to the outputs based on a black-box approach.

The vapour compression cycle model, named innerCycle, then uses this partial model as a re-

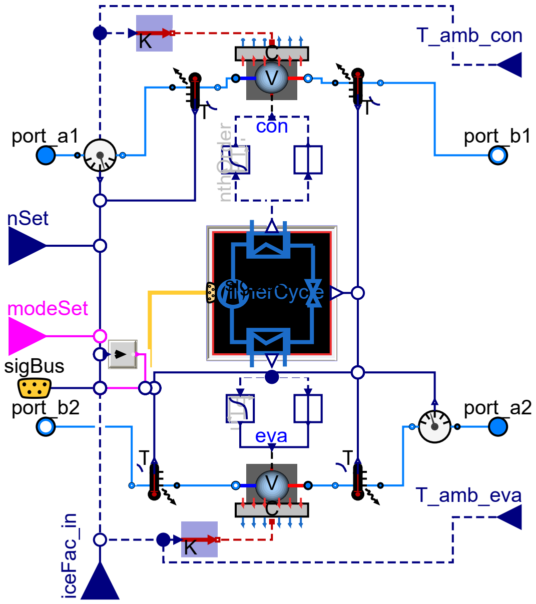


Figure 1. Partial model of a reversible thermal machine

placeable option. Based on the Boolean input `modeSet`, the cycle switches between data for cooling and heating. This approach is visualized in Figure 2. At the current state, we have introduced three performance data approaches.

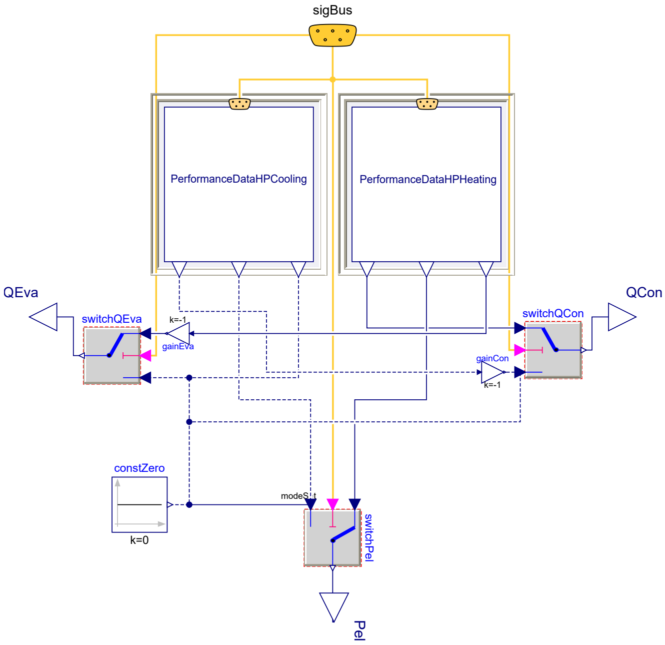


Figure 2. Black-box model of the reversible vapour compression cycle, exemplified for a HP

1. *Functional approach*: Any modelica function using variables in Table 1 to calculate the outputs.

2. *2D-Data*: All VCMs on the market require data according to the EN 14511 (EN 14511-1:2018-03 2018). As table inputs, $T_{\text{Eva,in}}$ and $T_{\text{Con,out}}$ are regarded. For non fixed-speed devices, the compressor speed is necessary to model the part load behaviour. We account for part load by linearly multiplying model outputs with `nSet`. Note however, that table data was not necessarily obtained at full load. Hence, we introduce a third approach.

3. *3D-Data*: Also based on EN 14511-1:2018-03 (2018), we add compressor speed as a table dependency. While this solves the part load issue, only some manufacturer publish tables this detailed.

All the above-mentioned approaches include two further options. First, a scaling factor may be used to scale the VCM according to demand. Second, we model the VCM internal effect of possible evaporator frosting. As we chose a black-box approach for the vapour compression cycle, gray-box modeling of frost is challenging. Hence, we adapt a black-box approach as well. The icing factor f_{ice} accounts for losses in evaporator efficiency due to frosting effects. This factor has to be calculated using external models, e.g. according to the COP correction proposed by Afjei and Wetter (1997). For output of the inner cycle it follows:

$$\dot{Q}_{\text{Eva,out}} = f_{\text{ice}} \cdot \dot{Q}_{\text{Eva}} \tag{2}$$

$$P_{\text{el,out}} = P_{\text{el}} \tag{3}$$

$$\dot{Q}_{\text{Con,out}} = \dot{Q}_{\text{Eva,out}} + P_{\text{el,out}} \tag{4}$$

3.2 Inertia

As stated in the prior section, the `innerCycle` model is based on stationary data points. Hence, transient calculation of the performance in the vapour compression cycle does not take place. This is especially problematic during on-off switching of the device. When the compressor is switched on, it requires instantaneous electrical power. However, the refrigerant and components have mass and thermal inertias. As a result, the effective heat flow is delayed. Thus, we consider this inertia of the inner cycle by adding a `CriticalDamping` filter. This adds an PT delay of n-th order on the black-box outputs $\dot{Q}_{\text{Con}}$ and $\dot{Q}_{\text{Eva}}$. Again, usage and order of the filter are custom parameters.

3.3 Heat exchangers

The heat flow rates calculated in the `InnerCyle` are directly fed into the secondary fluids. Current open-source models neglect heat losses with the ambient. However, the dynamics during heat up and cool down phases may be relevant for the system interaction. Thus, we extend the heat exchanger `HeaterCooler_u` from the `IBPSA`. We add a capacity between secondary fluid and ambient air, refer to Figure 3. Thus, the ambient temperature influences the secondary fluids. Heating the device up takes longer due to higher capacity. Similarly, the capacity may transfer heat to the secondary fluid during cool down, even though the device is turned off. As some black-box data implicitly accounts for such losses, not all use cases require this option. It's usage is thus optional.

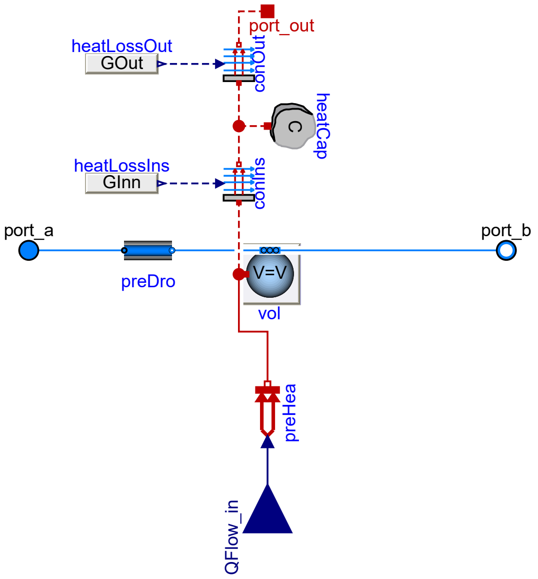


Figure 3. Model for evaporator and condenser

3.4 Safety controls

In section 2 we point out that current model approaches neglect interaction of system controls with the internal safety control of VCMs. As only the interaction with the system is relevant to us, we regard all safety controls which influence the long term performance of the system level as relevant. Furthermore, the order of safety controls is relevant in Modelica. We choose the following order, going from set point generators over logical controls to physical safety controls.

- **Thermal disinfection:** Frequent thermal disinfection of domestic hot water is required for HPs to ensure safety of users. Temperatures over 60 °C are necessary (Van Kenhove 2018).

- **Defrost:** For air-source VCMs, evaporators may need a defrosting control to ensure an operation without frost on the coils. For reversible VCMs, we model reverse cycle defrost. If the VCM is not reversible, defrost is modeled using auxiliary heaters.

- **On/off control:** Most VCMs require a minimal run time, a minimal off-time and forbid frequent on/off switching. All three requirements depend on `nSet` and `pre(nSet)`. Thus, they are included in one model. Switching single requirements on or off is achieved through booleans.

- **Operational envelope:** The compressor of VCMs can only operate in a given operational envelope. If outside of the envelope, pressure switches and thermal resistors shut the VCM off. In the model, we select `TConOutMea` and `TEvaInMea` as input for the model. As in Cimmino and Wetter (2017), a hysteresis prevents frequent switching.

- **Frost protection:** For water or brine sources, the fluid temperature may not fall below certain thresholds to prevent phase change. Hence, we add a safety control to ensure `TEvaOutMea` and `TConInMea` are greater than a given threshold.

All controls aggregated, the resulting `SafetyController` is depicted in Figure 4. Each option may be disabled. To better analyze simulation results, integer outputs show users the number of times the control was used.

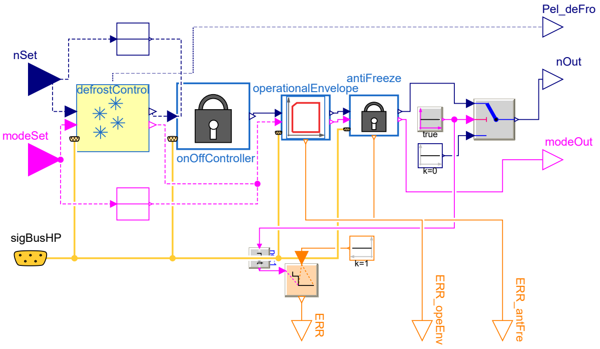


Figure 4. Implemented safety controller for the VCM

3.5 System integration

Lastly, we adapt a model of the overall VCM system by adding additional components used in VCM. These include secondary pumps, fans and an auxiliary heater. Again, all listed components are replaceable and optional. Not optional is the system controller. It includes the safety controls and the control to generate the signals `nSet` and `modeSet`.

This system model is currently implemented for a heat pump. An example is located in the package `AixLib.Systems.HeatPumpSystems`.

For reference, the models may be viewed and tested on the latest commit `38c19751b` in the repository `AixLib`[1].

4 Heat pump calibration

To show validity and modularity of our presented approach, we calibrate a brine/water HP with different black-box approaches. As calibration method, we select the approach presented in Mehrfeld, Nürenberg, and Dirk Müller (2021).

Experiments are performed at a Hardware-in-the-Loop test bench. Temperature inputs, outputs and consumed $\hat{P}_{\mathrm{el}}$ are presented in Figure 5. Measured data is denoted using the hat operator, e.g. $\hat{T}$. The data between 6000 and 8000 s data is not used for calibration but for validation only. We calibrate the *NRMSE*, refer to Equation 5, of both P_{el} and $T_{\mathrm{Con,out}}$ weighted equally.

$$NRMSE = \frac{1}{\hat{x}_{\max} - \hat{x}_{\min}} \cdot \sqrt{\frac{\sum_{i=1}^{n}(x_i - \hat{x}_i)^2}{n}} \qquad (5)$$

[1] `https://github.com/RWTH-EBC/AixLib`

Table 2. Data of the HP on the test bench according to EN 14511

P_{el} in W		
$T_{Eva,in}$ in °C / $T_{Con,out}$ in °C	0	10
35	1300	1500
55	1900	2300
$\dot{Q}_{Con}$ in W		
$T_{Eva,in}$ in °C / $T_{Con,out}$ in °C	0	10
35	6100	8400
55	5700	7600

As calibration parameters, we select only those which are most sensitive to our target values. These parameters are mass flow rate at condenser $\dot{m}_{Con}$, volume of condenser V_{Con} and frequency of the inertia filter $f_{inertia}$. First, we model the HP according to manufacturer data listed in standard EN 14511 (EN 14511-1:2018-03 2018). The data is stated in Table 2. Second, we use a Carnot approach with a constant quality grade. The consumed electrical power is constant at nominal conditions. This approach is equivalant to the Carnot approach in the `IBPSA` for on/off heat pumps. The equations are as follows:

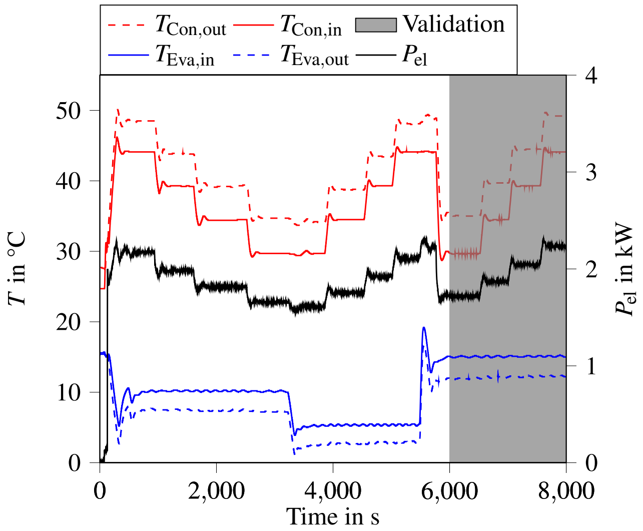


Figure 5. Experimental data for the calibration and validation

$$P_{el} = P_{el,Nom} \tag{6}$$

$$\dot{Q}_{Con} = P_{el,Nom} \cdot \xi \cdot \frac{T_{Con,out}}{T_{Con,out} - T_{Eva,in}} \tag{7}$$

In this study, the quality grade ξ and nominal power $P_{el,Nom}$ are additional calibration parameters.

4.1 Results and Discussion

The resulting parameter values of both calibrations are listed in Table 3 together with applied boundaries. Additionally, time series plots for both cases are displayed in Figure 6.

Table 3. Applied calibration parameters and results

Parameter	Min	Max	Results	
			Table	Carnot
$\dot{m}_{Con}$ in $\mathrm{kg\,s^{-1}}$	0.2	0.6	0.404	0.214
V_{Con} in L	1	50	4.47	1.59
$f_{inertia}$ in µHz	1	300	12	13.2
ξ in %	5	50	-	43.18
$P_{el,Nom}$ in kW	1.5	2.5	-	1.88
NRMSE Calibration			0.046	0.053
NRMSE Validation			0.041	0.055

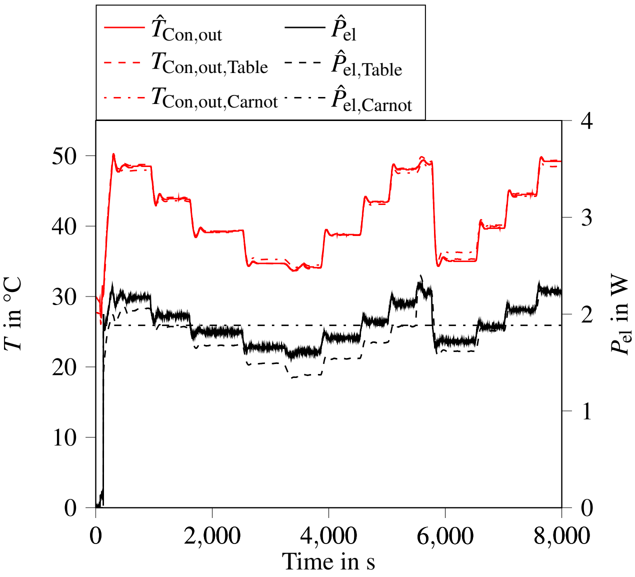


Figure 6. Time series plots for objectives of calibration.

First of all, both model approaches are able to reproduce the measured temperature. This is possible due to the dynamics introduced by the additional inertia and volume of the heat exchanger. For the validation part, source temperature increases and thus induces an extrapolation of the black-box approach. Hence, the error increases slightly.

However, the black-box is based on just four points (Table) or two parameters (Carnot). As interpolation typically yields sufficient accuracy, both approaches may be used for typical operation limits for brine/water HPs. Looking at the electrical power, both approaches differ. Obviously, the constant $P_{el,Nom}$ leads to error prone model outputs. Not as obvious, the measured data according to EN 14511-1:2018-03 (2018) also shows discrepancies. Most interestingly, the error is smaller during validation. As $T_{Con,out,Table}$ is higher than $\hat{T}_{Con,out}$ during validation, simulated $P_{el,Table}$ increases as well. Note that the results of the calibration depend on the weighting of objectives. With a higher weighting towards P_{el}, the fit would increase.

Concluding, both approaches may be used based on the aim of the simulation analysis. Minimal data input of just four data points or two calibrated parameters yield accurate representations when compared to experimental data.

5 Seasonal building performance simulations

The main goal of our contribution is to develop a modular model for the use in energy systems. Hence, our second use case concerns the integration of said model into a building energy system.

The building is based on the high-order approach in the `AixLib` (D. Müller et al. 2016). Demand of domestic hot water is neglected.

An air/water HP is simulated, hence the anti-freeze protection function is not applied. The simulation period is equal to one heating period, which is defined from 1st October to the 30th April. Besides computation time and the number of state events, the SCOP is calculated.

Model parameters are selected based on calibration results, scaled to the manufacturer data of an exemplary fixed-speed HP. Thermal disinfection is performed once a week by increasing the storage tank temperature above 60 °C for a duration of 15 min. If the outdoor temperature falls below 5 °C, defrost cycles are modeled by a daily 10 min reversal of the refrigeration circuit.

To demonstrate influences of introduced models, we analyze eight model configurations (MC). Together with relevant results, they are listed in Table 4. The simulation settings are stated in Table 5

Table 4. Definition and results of model configurations. Usage of option is indicated by (x).

Configuration	Heat losses	Additional inertias	Operational envelope	On/Off control	Thermal disinfection	Defrost	Computation time in h	state events in thousand	*SCOP*
1	o	o	o	o	o	o	5.2	46.1	2.44
2	x	x	o	o	o	o	6.9	45.7	2.21
3	x	o	x	x	o	o	6.5	64.8	2.14
4	o	x	x	x	o	o	7.2	61.8	2.36
5	x	x	x	x	o	o	7.4	64.7	2.15
6	x	x	x	x	x	o	7.5	65.3	2.14
7	x	x	x	x	o	x	7.8	65.6	1.88
8	x	x	x	x	x	x	7.8	66.0	1.88

5.1 Results and Discussion

Looking at Table 4, we want to analyze two influences in detail.

Table 5. Simulation settings for all studied configurations

Setting	Value
Start time	273 days
Stop time	485 days
Interval length	5 min
Solver	Dassl
Tolerance	0.0001
Store variables at events	true

First, the computation time in MC_2 increased by a factor of 1.33. As both heat losses and inertias are active in MC_2, the direct impact of both options is not distinguishable. By comparing MC_3 and MC_4 to MC_5, the impact can be assessed. Neglecting the system inertias in MC_3 leads to a 12.6 % reduction in computation time compared to MC_5. If the heat losses are not simulated, the computation time decreases by 3.6 % compared to MC_5. Despite increased number of state events, MC_3 is faster than MC_2. Overall, the `CriticalDamping` block of third order for mapping the inertia increases the computation time without having a noteworthy impact on the SCOP. This increase may be attributed in six additional continuous time states introduced by the model option. The usage of heat losses only introduce two new continuous time states. In general, the usage of inertias should always be weighted against the increased computation time.

Second, we analyze the influence of safety controls on the computation time and SCOP. Both the computation time and the number of state events increase between MC_1 and MC_3 to MC_8. The compliance with the operating envelope and on/off control evokes 15000-20000 state events depending on the MC, resulting in an increase of the computation time by at least 7.5 %. The operating envelope is not exceeded at any time. Therefore, this increase is due to compliance with the switching cycles. In MC_6, the influence of thermal disinfection is analyzed. Compared to MC_5, the SCOP is decreased by 0.5 %, and the computation time is increased by 1.2 %. The chosen period of 15 min as well as the week-based circuit are possible reasons for the small influence. Overall, the defrost control has the greatest influence on the SCOP. As a result of reversing the refrigeration circuit, the SCOP decreases by 12.6 %. On the one hand, this is due to the fact that the table data already take defrosting losses into account. On the other hand, the duration and timing of the defrosting cycles in the use case are not based on validated assumptions. Following defrost control, the heat losses show the highest impact on the SCOP. Comparing MC_1 to MC_2, the SCOP decreases by 9.4 %. To show that this decrease is not induced by the additional inertias, we compare MC_4 to MC_5. Here, the decrease amounts for 8.9 %. This is explained by the fact that inertias change the dynamics of heat flow and not the quantity itself.

Generally, the obtained SCOP may seem small. However, we examined a supply temperature of 55 °C using a 2 K bandwidth hysteresis control. For these systems, the SCOPs are in similar ranges compared to SCOPs obtained in field tests (Huchtemann and Dirk Müller 2012).

6 Conclusion

A modular model for reversible heat pumps and chillers is presented. The vapour compression cycle is modeled as a replaceable black-box. Contrary, relevant interactions with the whole energy system are modeled as gray-box. Most notably, safety controls such as on/off controls and the operational envelope are regarded in the model. All models are optional, enabling users to choose modeling depth based on their simulation aim.

To demonstrate validity of our approach, we calibrate two different black-box approaches onto measured data. While both manage to reproduce supply temperatures, the Carnot based approach fails to match electricity demands.

Besides calibration, we show influences of different model options onto simulation speed and efficiency of heat pumps. The presented results indicate that safety controls should always be regarded neglected on system integration level.

While the presented approach is reversible, our background focuses mainly on the use of VCM as a heat pump in building energy systems. Calibration and validation of chiller applications may show limitations of the approach. Especially validation of experiments using a four-way reversing valve are of interest. As we neglect internal effects of the VCM, losses or dynamics during reverse cycle mode may lead to poor model accuracy.

Last but not least, we already implemented various black-box approaches in the `AixLib` (D. Müller et al. 2016). Researchers and practitioners are invited to contribute new black-box approaches to this open-source modeling library. Thus, the variety of simulation aims covered by the presented modeling approach may increase, yielding a more thorough understanding of heat pumps and chillers in system applications.

Acknowledgements

References

Afjei, Thomas and Ralf Dott (2011). "Heat pump modelling for annual performance, design and new technologies". In: *12th Conference of International Building Performance Simulation Association*, pp. 1–8.

Afjei, Thomas and Michael Wetter (1997). *Compressor heat pump including frost and cycle losses*. URL: https://simulationresearch.lbl.gov/wetter/download/type204_hp.pdf.

Carbonell Sánchez, Daniel et al. (2012). "Numerical analysis of heat pumps models: comparative study between equation-fit and refrigerant cycle based models". In: *Solar energy for a brighter future: book of proceedings: EuroSun 2012*.

Chamoun, Marwan, Romuald Rulliere, Philippe Haberschill, and Jean Francois Berail (2012). "Dynamic model of an industrial heat pump using water as refrigerant". In: *International Journal of Refrigeration* 35.4, pp. 1080–1091. ISSN: 0140-7007. DOI: https://doi.org/10.1016/j.ijrefrig.2011.12.007. URL: https://www.sciencedirect.com/science/article/pii/S0140700711003082.

Chamoun, Marwan, Romuald Rulliere, Philippe Haberschill, and Jean-Louis Peureux (2013). "Modelica-based modeling and simulation of a twin screw compressor for heat pump applications". In: *Applied Thermal Engineering* 58.1, pp. 479–489. ISSN: 1359-4311. DOI: https://doi.org/10.1016/j.applthermaleng.2013.04.020. URL: https://www.sciencedirect.com/science/article/pii/S1359431113002901.

Cimmino, Massimo and Michael Wetter (2017-07). "Modelling of Heat Pumps with Calibrated Parameters Based on Manufacturer Data". en. In: pp. 219–226. DOI: 10.3384/ecp17132219. URL: https://ep.liu.se/en/conference-article.aspx?series=ecp&issue=132&Article_No=22 (visited on 2021-04-17).

De Coninck, Roel et al. (2010). "Modelling and simulation of a grid connected photovoltaic heat pump system with thermal energy storage using Modelica". In: *8th international conference on system simulation*, p. 21.

Dechesne, Bertrand et al. (2017). "Comparison of a dynamic model and experimental results of a residential heat pump with vapor injection and variable speed scroll compressor". In: *The 30th International Conference on Efficiency, Cost, Optimization, Simulation and Environmental Impact of Energy Systems*.

EN 12900:2013-10 (2013-10). *Refrigerant compressors - Rating conditions, tolerances and presentation of manufacturer's performance data*. Tech. rep.

EN 14511-1:2018-03 (2018-03-14). *Air Conditioners, Liquid Chilling Packages and Heat Pumps for Space Heating and Cooling and Process Chillers, with Electrically Driven Compressors - Part 1: Terms and Definitions*. Tech. rep. Bruxelles, Belgium: CEN/TC 113. URL: https://standards.cen.eu/dyn/www/f?p=204:110:0::::FSP_PROJECT:59177&cs=18B95E474391A98AD6EAB83913BAE7714.

Heat Roadmap Europe (2017). *Heating and Cooling - facts and figures*. URL: https://www.isi.fraunhofer.de/content/dam/isi/dokumente/cce/2017/29882_Brochure_Heating-and-Cooling_web.pdf.

Huchtemann, Kristian and Dirk Müller (2012). "Evaluation of a field test with retrofit heat pumps". In: *Building and Environment* 53, pp. 100–106. ISSN: 03601323. DOI: 10.1016/j.buildenv.2012.01.013.

IEA (2020). *Heat Pumps*. URL: https://www.iea.org/reports/heat-pumps.

International Building Performance Simulation Association (2018). *IBPSA Project 1: BIM/GIS and Modelica Framework for building and community energy system design and operation*. URL: https://ibpsa.github.io/project1/.

Jin, Hui (2002). "Parameter estimation based models of water source heat pumps". Dissertation. Shanghai: Jiaotong University.

Jorissen, Filip et al. (2018). "Implementation and Verification of the IDEAS Building Energy Simulation Library". In: *Journal of Building Performance Simulation* 11 (6), pp. 669–688. DOI: 10.1080/19401493.2018.1428361.

Mehrfeld, Philipp, Markus Nürenberg, Martin Knorr, et al. (2020). "Dynamic evaluations of heat pump and micro combined heat and power systems using the hardware-in-the-loop approach". In: *Journal of Building Engineering* 28, p. 101032.

Mehrfeld, Philipp, Markus Nürenberg, and Dirk Müller (2021). "Model Calibration of an Air Source Heat Pump System for Transient Simulations in Modelica". en. In: *13th IEA Heat Pump Conference*. Jeju, Korea, p. 11.

Müller, D. et al. (2016). "AixLib – An Open-Source Modelica Library within the IEA-EBC Annex 60 Framework". In: *BauSIM 2016*.

Van Kenhove, Elisa (2018). "Coupled Thermohydraulic and Biologic Modelling of Legionella Pneumophila Proliferation in Domestic Hot Water Systems". PhD Thesis. Gent: University Gent.

VDI 3633:2014-12 (2014-12). *Simulation of systems in materials handling, logistics and production - Fundamentals*. Tech. rep.

Wystrcil, Dominik and Doreen Kalz (2012). "Thermohydraulische Modellierung eines Niedrigexergiesystems zur Gebäudeheizung- und kühlung und exergetische Bewertung von Regelungsstrategien". In: *Fourth German-Austrian IBPSA Conference*.

Modelica Modeling and Simulation for a Micro Gas-Cooled Reactor

Huimin Zhang[1] Erhui Chen[1] Yangyang Liang[1] Li Wang[1] Jun Wang[1] Shuhong Du[1]
Liping Chen[2] Fanli Zhou[2] Ji Ding[2] Haiming Zhang[2]
[1]China Nuclear Power Engineering Co., Ltd., Beijing, China,
{zhanghm, cheneh, liangyya, wanglie, wangjuna, dush}@cnpe.cc
[2]Suzhou Tongyuan Software & Control Technology Co., Ltd. Suzhou, China, {chenlp, zhoufl, dingj} zhanghm}@tongyuan.cc

Abstract

Highly compact micro nuclear reactors which have broad energy advantages in the application of ocean, land, space, and sky, become a hot research topic in the international nuclear industry recently. In this paper, Modelica language was used in the system modeling and simulation of a micro gas-cooled reactor. The Modelica model was self-developed by China Nuclear Power Engineering Company and the MWorks developed by Suzhou Tongyuan was chosen as the simulation platform. Two simulations of a concept micro gas-cooled reactor design were carried out. One is the extreme accident scenario and the other is a normal load-following operation. The simulation results showed that the reactor has good inherent safety even under the extreme accident, in which the reactor shutdown can be achieved only by the negative reactivity result from the increase of core temperature and the fuels were not damaged since the decay heat was removed by passive air cooling from outside of the reactor pressure vessel. The reactor also has good load-following performance, which can be achieved by simply adjusting the helium inventory (or pressure) and the control rod position, while the core temperature and power generation efficiency kept constant.

Keywords: system modeling, system simulation, micro-nuclear reactor, load-following

1 Introduction

The advanced micro nuclear energy system is a specific nuclear energy technology with high flexibility and sustainable and reliable energy supply, and has a good application prospect in remote areas, islands and other places with poor traffic and difficult energy supply. The micro nuclear energy system with the corresponding facilities can achieve the stable supply of clean energy, and has broad energy advantages in the application of ocean, land, space, and sky.

The concept micro gas-cooled reactor researched in this paper is a typical micro nuclear energy system with complex structures and multi-disciplinary, such as neutron physics, thermal engineering, energy conversion, electricity and control. Compared with a large-scale nuclear plant which adopt the traditional ‘divide and conquer’ design concept and is composed of a large amount of fully decoupled subsystems, the compact micro nuclear reactor has fewer subsystems but its subsystems are tightly coupled due to the constraints of the volume and weight. To handle the complexity aroused from this coupling issue and to better predict the reactor dynamic behavior, it is necessary to perform the system simulation across multiple disciplines and domains. Modelica is a unified modeling language for complex physics systems with multi-disciplinary. Oak Ridge National Laboratory (ORNL) adopted Dymola platform based on Modelica to establish the reactor model library TRANSFORM for the system modeling and simulation of high temperature gas-cooled reactor (HTGR) (Hale et al. 2015), nuclear thermal propulsion rocket(Rader et al. 2019)and molten salt reactor(Greenwood 2018; Greenwood et al. 2018). In China, MWorks platform based on Modelica developed by Suzhou Tongyuan was used for the integrated system simulation of manned spacecraft (Bainan et al. 2020) and two-phase flow (Yanping et al. 2021). In this paper, the MWorks platform was used for the system modeling and simulation of the concept micro gas-cooled reactor.

The whole system of the micro gas-cooled reactor is shown in Figure 1. The direct Brayton cycle is used for the heat-work conversion. As the medium of heat transfer and work, the helium is heated by the reactor core and then enters the turbine to expand, and then enters the recuperator to reduce the temperature. After further cooling by the precooler, the helium flows across the two-stage compressor to increase the pressure. Finally, the helium is heated by the recuperator and returns to the reactor core to repeat

DOI
10.3384/ecp21181569

the thermal cycle process. The compressors, turbine and generator are connected by a main shaft, and the turbine provides torque power to drive the generator and compressors to rotate.

As shown in Figure 2, the reactor core is composed of 36 fuel bricks, an inner reflector assembly, an outer reflector, a core barrel and 12 control drums. Graphite is used as both the neutron moderator and the fuel structure, and is the major material of the fuel bricks, reflectors and control drums. The coated fuel particles TRISO (the design temperature limit is 1600℃ (Yang et al. 2010)) are dispersed in the fuel bricks. The coolant channels are distributed regularly inside the fuel blocks as holes.

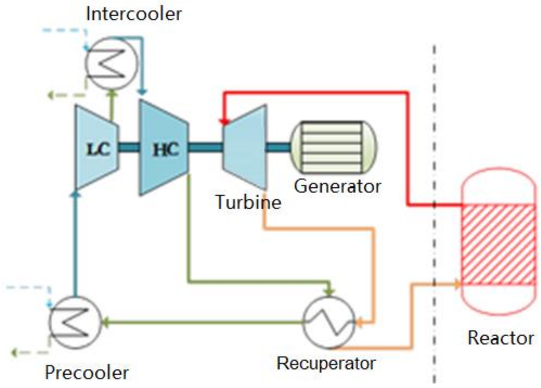


Figure 1. Main system of the micro gas-cooled reactor.

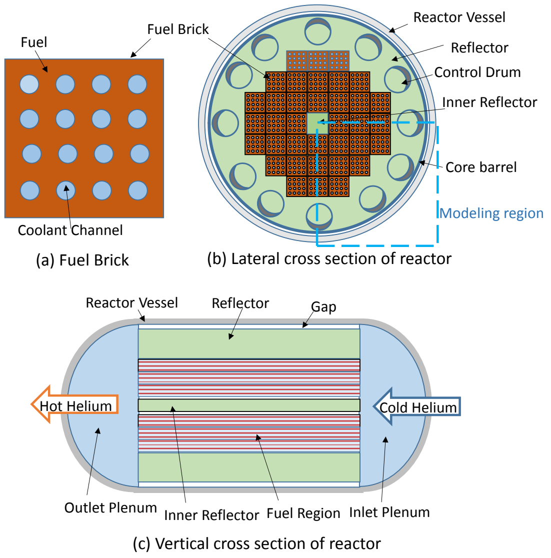


Figure 2. Sketch of reactor structure.

2 Modelica model

The Modelica model of the micro gas-cooled reactor system was established, as shown in Figure 3. The model consists of five subsystems and the data exchange is accomplished by Modelica interface. The five subsystems are described as follows:

(1) Reactor system. The core heat generated by the reactor system provides the energy source for the Brayton cycle. The reactor system focuses on the core reactivity, nuclear power, thermal-hydraulic, decay heat power and residual heat removal power. The system includes the models of the point reactor, decay heat power and thermal-hydraulic.

(2) Heat engine system. The heat engine system focuses on the Brayton cycle to realize the heat-work conversion. The system includes the models of turbine, compressor, regenerator, precooler, intercooler, and pipeline.

(3) Heat sink system. The heat sink system provides the cold source for the precooler and intercooler in the heat engine system, and is simply realized by mass inlet boundary.

(4) Electricity system. The electricity system converts the rotational kinetic energy in the heat engine system into the electricity energy, and is simply realized by a given load.

(5) Control system. The control system focuses on the control of the power operation and heat sink.

The models of the reactor system, heat engine system and control system will be discussed detail below.

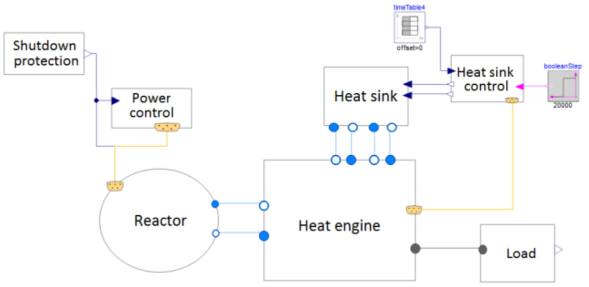


Figure 3. Modelica model of the micro gas-cooled reactor system.

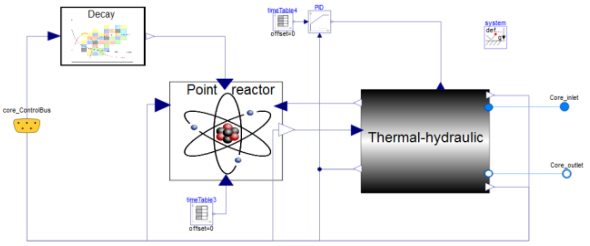


Figure 4. Modelica model of the reactor model.

2.1 Reactor system

The Modelica model of the reactor system is shown in Figure 4, including the models of the point reactor, decay heat and thermal-hydraulic. The decay heat model is implemented by calling external C functions by virtue of the external function interface in MWorks. The helium inlet and outlet in the reactor system are connected to the helium outlet and inlet in the heat engine system, respectively.

2.1.1 Point reactor model

The reactor neutron model is based on the point reactor neutron dynamics. The three-dimensional effect of the neutrons space dynamics is neglected, and the neutron flux distribution is fixed in space and only change with time. The equations are listed as follows:

$$\frac{dN}{dt} = \frac{\rho - \beta}{\Lambda} N + \sum_{i=1}^{6} \lambda_i C_i \tag{1}$$

$$\frac{dC_i}{dt} = \frac{\beta_i}{\Lambda} N - \lambda_i C_i \tag{2}$$

$$\beta = \sum_{i=1}^{6} \beta_i \tag{3}$$

$$\rho = \rho_{ini} + \rho_T + \rho_{\mathrm{Xe}} + \rho_{ext} \tag{4}$$

where N is the average neutron density, Λ is the neutron generation time, β is the total delayed neutron fraction, β_i is the delayed neutron fraction of group i, C_i is the precursor concentration of the delayed neutron of group i, λ_i is the decay constant of the delayed neutron of group i, and ρ is the net reactivity. The initial reactivity (ρ_{ini}), temperature reactivity (ρ_T), xenon reactivity (ρ_{Xe}) and external reactivity (ρ_{ext}) introduced by control rods are considered in the model.

2.1.2 Thermal-hydraulic model

Thanks to the symmetry, one quarter of the whole reactor is chosen in the thermal-hydraulic (T-H) model as shown in Figure 5, including the fuel bricks, inlet and outlet plenums, and thermal components (reflectors, core barrel and pressure vessel). Since the major material of the control drums is same as the reflector, they are simply considered same and merged into the reflector component in the T-H modeling.

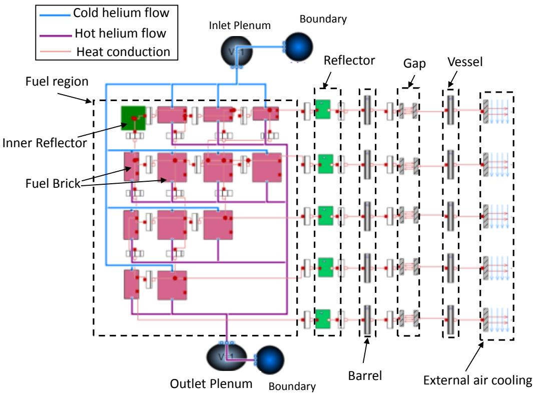
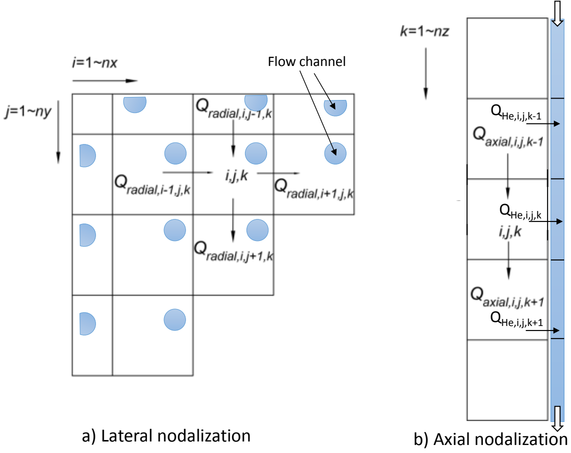


Figure 5. Modelica model of the reactor thermal-hydraulic.

Since there is an amount of graphite inside core which is thermal conductor, the heat conduction phenomenon in the solid core regions is very important and need to be modeled. In reality, gaps may exist between the bricks, deteriorate the heat transfer. At this preliminary work, the assumption of no gap is taken and the heat conduction are ideally considered between the fuel bricks, reflectors and barrel.

Because the nuclear power in different fuel bricks are not uniform, the heat transfer phenomenon in the reactor region have the 3-dimentional behavior, which should be taken into account in the modeling and simulation. In this paper, a T-H model with coarse 3-D nodalization is established by using the lumped parameter method.

The nodalization of the fuel region is shown in Figure 6. There are 13 lateral nodes each representing a brick, and the fuel bricks are divided into 5 segments axially. Therefore there are 65 blocks, of each contains a solid fuel block and a flow segment. Each solid fuel block has a heat source which receives the nuclear heat power, transfers heat energy to the neighbor solid fuel blocks vertically and laterally by thermal conduction, and releases heat energy into the gas segment inside it by convection.



a) Lateral nodalization

b) Axial nodalization

Figure 6. Nodalization of fuel region.

The energy function of the solid block is:

$$\begin{aligned}&\rho_{f,i,j,k} V_{f,i,j,k} C_{p,i,j,k} \frac{dT_{block,i,j,k}}{dt} \\ &= Q_{nuclear,i,j,k} + Q_{radial,i-1,j,k} + Q_{radial,i,j-1,k} + Q_{axial,i,j,k\text{-}1} \\ &- Q_{\mathrm{He},i,j,k} - Q_{radial,i+1,j,k} - Q_{radial,i,j+1,k} - Q_{axial,i,j,k+1}\end{aligned} \tag{5}$$

where i, j, k represent the coordinates, V_f is the segment volume, ρ_f is the density, C_p is the specific heat capacity, T_{block} is the temperature, $Q_{nuclear}$ is the input nuclear energy, Q_{He} is the output heat convection energy with helium, Q_{radial} is the heat conduction energy with the radial segments of the surrounding fuel assembly, Q_{axial} is the heat conduction energy with the axial segments within the fuel assembly.

Since no cross flow between flow channels, a flow channel can be modeled as a pipe using the Modelica standard library. The helium compression is considered and the heat convection is calculated as (Hale et al. 2015):

$$Q_{\mathrm{He}} = h_{\mathrm{He}} A_{\mathrm{He}} (T_{block} - T_{\mathrm{He}}) \tag{6}$$

$$h_{\mathrm{He}} = \frac{\lambda_{\mathrm{He}}}{D_{channel}} \mathrm{Nu}_{\mathrm{He}} \tag{7}$$

$$\mathrm{Nu}_{\mathrm{He}} = \begin{cases} 2\left(\dfrac{m_{\mathrm{He}} C_{p,\mathrm{He}}}{\lambda_{\mathrm{He}} l}\right)^{0.333} & \mathrm{Re}_{\mathrm{He}} < \mathrm{Re}_{crtical} \\ 0.023\,\mathrm{Re}_{\mathrm{He}}^{0.8}\,\mathrm{Pr}_{\mathrm{He}}^{0.4} & \mathrm{Re}_{\mathrm{He}} \geq \mathrm{Re}_{crtical} \end{cases} \tag{8}$$

where Q_{He} is the heat convection energy, h_{He} is the heat convection coefficient, A_{He} is the heat transfer area, T_{He} is the helium temperature, λ_{He} is the conductivity, $c_{p,\mathrm{He}}$ is the specific heat capacity, $D_{channel}$ is the channel diameter, m_{He} is the helium mass flow rate, l is the channel length, $\mathrm{Nu}_{\mathrm{He}}$ is Nusselt number, $\mathrm{Pr}_{\mathrm{He}}$ is Prandtl number, $\mathrm{Re}_{\mathrm{He}}$ is Reynolds number, $\mathrm{Re}_{critical}$ is the critical Reynolds number between the laminar and turbulent.

The T-H modeling of the reflector and vessel are similar and simpler to the modeling of fuel region, also related to the heat conduction.

For the gap between the barrel and vessel, heat radiation is assumed to be the only heat transfer process. For the external vessel cooling, the air convection is simply modeled by setting a fixed convection heat transfer coefficient.

2.2 Heat engine system

As shown in Figure 7, the heat engine system adopts direct Brayton cycle to accomplish the heat-work conversion, and the system is composed of the turbine, compressors, recuperator, precooler, intercoolor, and pipes. The high pressure helium flows through the reactor core and gets heated, and then flows into the turbine to expand and does work to drive the main shaft to rotate. The rotating main shaft drives the compressors to compress the helium at the same time. The helium with relatively high temperature from the turbine goes through the lower pressure side of the recuperator and transfer heat to the high pressure helium which comes from the high pressure compressor. After that, the helium goes into the precooler to reduce the temperature and then enters the low pressure compressor. Then, the helium with low temperature and low pressure is compressed by the low and high compressors, which are connected by the intercooler, and thus the helium pressure is increased. Then the helium goes through the higher pressure side of the recuperator, and the temperature rises approximately to that at the turbine outlet. Finally, the helium flows into the reactor core and repeats such thermal cycle process again. The turbine model and compressor model is established to simulate the turbine and compressor. The heat exchanger model is established to simulate the recuperator, precooler, and intercooler. The working medium in the two sides of the precooler and intercooler are helium and cooling water, while the working medium in the two sides of the recuperator are both helium. The helium inlet and outlet in the heat engine system are connected to the helium outlet and inlet in the reactor system, respectively.

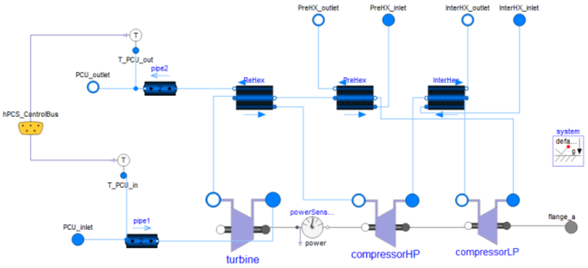


Figure 7. Modelica model of the heat engine system.

It is supposed that the helium flow is stable in the turbine and compressor, and the process is adiabatic. The ThermoPower library is used to establish the model of turbine and compressor. The characteristic curves, which are commonly used in the practical engineering, are used to describe the working process (Fern ándex-Villac éandPaniagua 2010).

The heat exchanger model is based on the models of pipe, heat components, heat conduction, interface and medium physical properties, as shown in Figure 8. The model of heat transfer and pressure drop in the pipe model are used to simulate the heat transfer and flow resistance between the fluid and pipe wall, and the heat conduction model is used to simulate the heat conduction between the cold and hot channel. The heat exchanger is divided into several segments, which are combined by the heat conduction model.

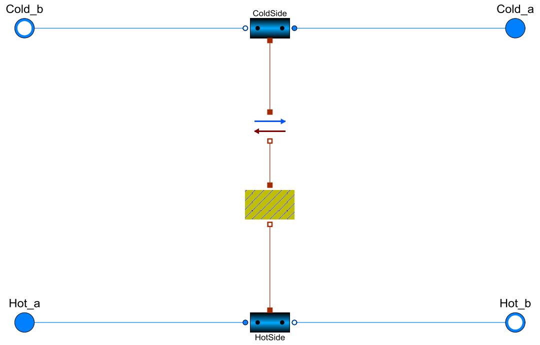


Figure 8. Modelica model of the heat exchanger.

2.3 Control system

The control system includes the power operation control and heat sink control in current research stage, as shown in Figure 9 and 10, respectively. The former controls the motion of control rods in the reactor system according to the monitored power and reactivity. The latter controls the helium flow rate in the precooler and intercooler in the heat sink system according to the helium temperature. The control system can be used to analyze the coupling matching of the reactor system and heat engine system, and it can also be used to explore the operation modes.

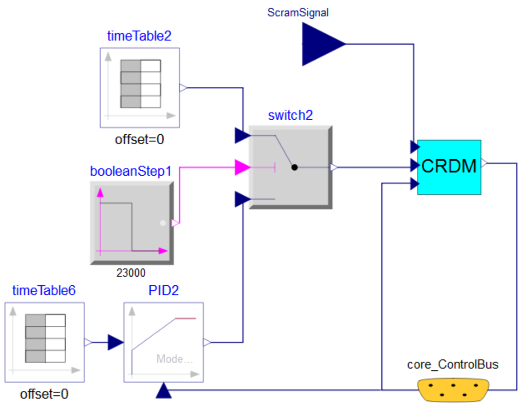


Figure 9. Modelica model of power operation control system.

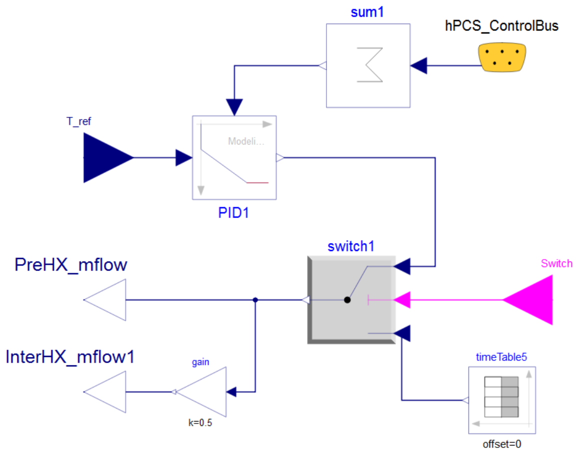


Figure 10. Modelica model of heat sink control system.

3 Calculation and analysis

Based on the above-mentioned theories and Modelica models, two simulations of the micro gas-cooled reactor design were carried out. One is the extreme accident scenario and the other is a normal reactor-engine load-following operation.

3.1 Extreme accident

The extreme accident is defined as such scenario, where the heat engines shut down and the helium flow rate drops to zero, with all of the control rods withdrew from the core. When the accident happens, the heat from the reactor core will transfer to the outer wall of the reactor pressure vessel, and then removed by passive air cooling in the environment. It is supposed that the accident happens after the reactor works at full power for 80 h. The various powers and average core temperature are shown in Figure 11 through Figure 13.

The control rods withdraw from core, introducing positive reactivity to the reactor core, and then the fission power rapidly rise up to 1.8 times of the full power in 80s. As a result, the core temperature rises promptly, leading to the rapidly increased negative temperature reactivity, and consequently the total reactivity decreases to a negative value. Therefore, the fission power drops to nearly zero, and the reactor shuts down. However, the total reactor thermal power is not zero because of the decay heat result from the continually generated fission products and activation products. With the residual heat removal, the reactor core temperature gradually goes down, leading to the decrease of negative temperature reactivity. Then the net reactivity increases and becomes positive after about 1.2 h of the accident, and thus the reactor achieves re-criticality. The final re-criticality power is about 5.2% of the full power, and the total thermal power which is the summation of the fission power and the decay heat power is equal to the residual heat removal power through the pressure vessel wall.

The maximum fuel temperature after the accident is about 1211 ℃ (The fuel and graphite is not separated, and the real maximum fuel temperature will be higher than the simulated value), which is lower than the design limit temperature (1600℃). Therefore, the reactor shutdown can be achieved after the accident only by the negative reactivity result from the increase of core temperature, and the reactor has good inherent safety. Tsinghua University has done the similar accident experiments on high temperature gas-cooled reactor-test module (HTR-10) reactor in 2003 (Gou et al. 2018), and the reactor shutdown was realized automatically by the negative reactivity result from the increase of core temperature. The safety of fuel and core can be guaranteed.

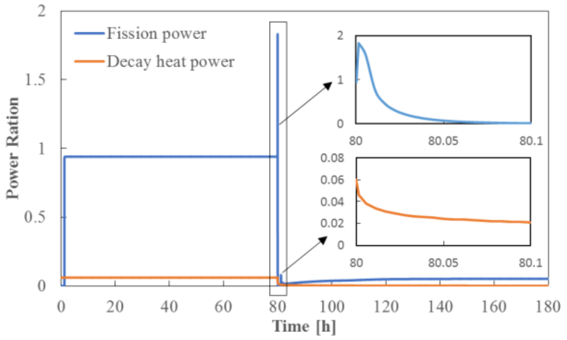


Figure 11. Nuclear fission power and decay heat power.

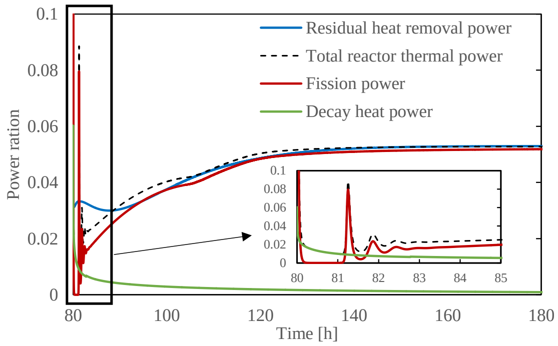


Figure 12. Reactor power and residual heat removal power.

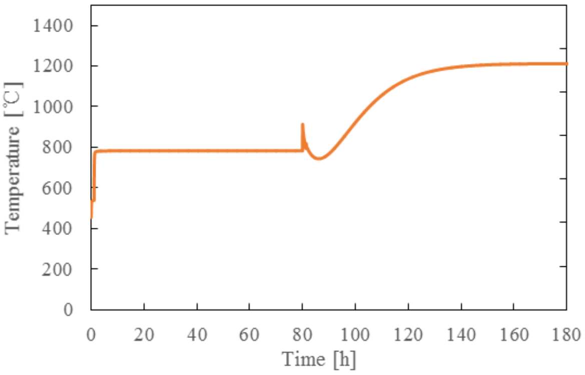


Figure 13. Core average temperature.

3.2 Reactor-engine load-following

It is supposed that the actual electricity load is time-dependent and fluctuates periodically. The fluctuation period is 24 h, which consists of peak period, steady period and low period. The demand power is determined by the electricity load, as shown in Figure 15. By simply adjusting the helium inventory (or pressure, setup in MWorks as shown in Figure 14) and the control rod position, the reactor nuclear power and net output power change periodically and synchronously, and the net output power coincides well with the demand power, which demonstrates that the function of load-following is realized successfully.

The reactor nuclear power and net output power are 4.10 MW and 1.37 MW respectively, during the peak period, and are 2.10 MW and 0.69 MW respectively, during the low period. The electric power generation efficiency is defined as the ratio of the net output power to the nuclear power, which changes little and nearly 33.0%, as shown in Figure 16.

As shown in Figure 17, the helium temperature at the core inlet and outlet are about 435℃ and 750℃, respectively, and both change periodically with a light fluctuation (±3℃).

As shown in Figure 18, the negative temperature reactivity changes little because of the nearly constant core temperature. The negative xenon reactivity changes periodically because of the periodically changed xenon concentration, which result from the fluctuating reactor power. Therefore, the external reactivity introduced by control rods also changes periodically to keep the reactor critical.

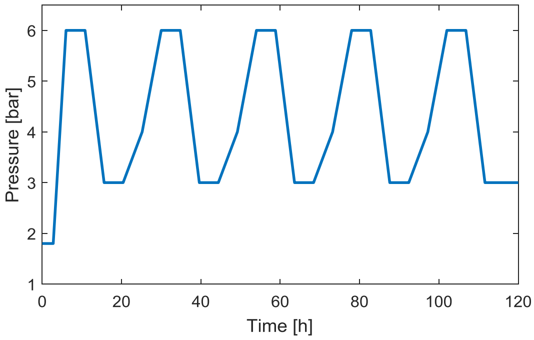


Figure 14. Helium pressure.

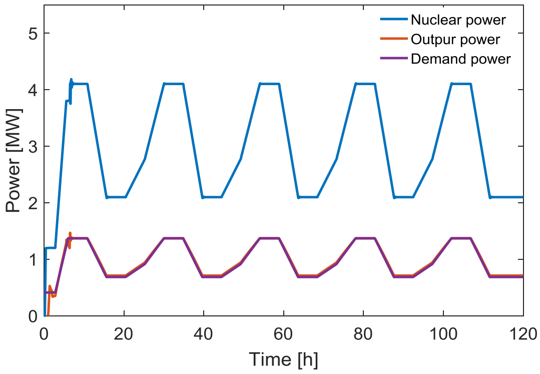


Figure 15. Nuclear power and output power.

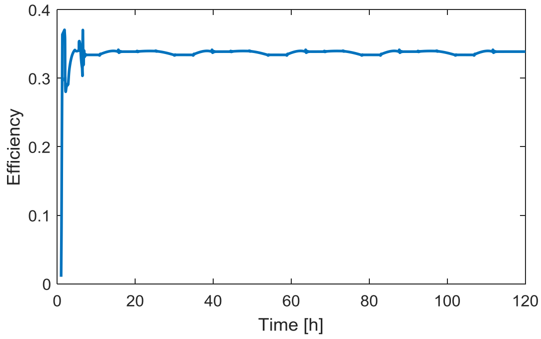


Figure 16. Power generation efficiency.

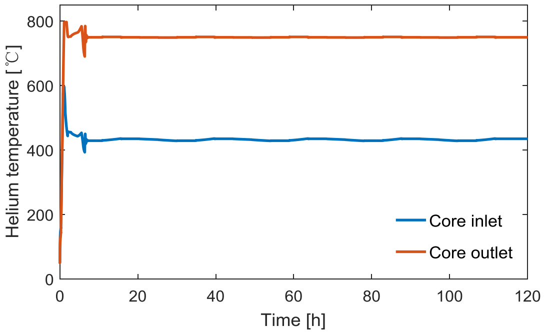


Figure 17. Helium temperature at core inlet and outlet.

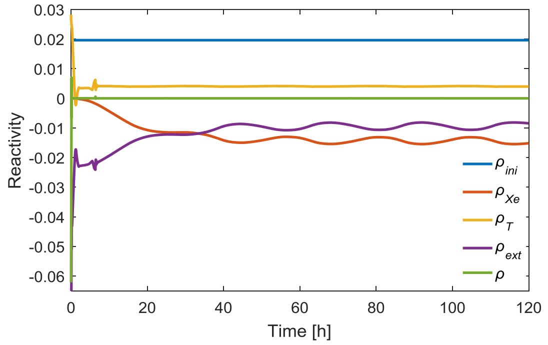


Figure 18. Core reactivity.

4 Conclusions

The system modeling and simulation for a micro gas-cooled reactor were carried out by Modelica language. The extreme accident and load-following were taken as examples to calculate and analyze. The simulation results showed that the reactor has good inherent safety even under the extreme accident, in which the reactor shutdown can be achieved only by the negative reactivity result from the increase of core temperature and the fuels were not damaged since the decay heat was removed by passive air cooling from outside of the reactor pressure vessel. The reactor also has good load-following performance, which can be achieved by simply adjusting the helium inventory (or pressure) and the control rod position, while the core temperature and power generation efficiency kept constant.

It should be mentioned that this Modelica model focus the system dynamic behavior, in which many simplifications have been taken. More verification will be carried out in future by comparing with the simulation results of high fidelity simulations or experiments. The simulation results is reasonable and accepted for the system dynamic analysis.

References

Bainan, Z., Faren, Q., Tao, X., Yang, L., & Wei, W. (2020). Model based development method of manned spacecraft: Research and practice[J]. *Acta Aeronauticaet Astronautica Sinica*, *41*(7), 23967-023967. DOI: https://doi.org/10.7527/s1000-6893.2020.23967

Fern ández-Villac é, V., & Paniagua, G. (2010). "Simulation of a combined cycle for high speed propulsion". Paper presented at the 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition. DOI: https://doi.org/10.2514/6.2010-1125

Gou, F., Liu, Y., Chen, F.-B., & Dong, Y.-J. (2018). Thermal behavior of the HTR-10 under combined PLOFC and ATWS condition initiated by unscrammed control rod withdrawal. *Nuclear Science and Techniques*, *29*(9), 1-9. DOI: https://doi.org/10.1007/s41365-018-0472-3

Greenwood, M. S. (2018). "Molten Salt-Fueled Nuclear Reactor Model for Licensing and Safeguards Investigations" (1650-3686). URL: https://www.osti.gov/servlets/purl/1509585

Greenwood, M. S., Betzler, B. R., & Qualls, A. L. (2018). "Dynamic System Models for Informing Licensing and Safeguards Investigations of Molten Salt Reactors". URL: https://www.osti.gov/servlets/purl/1456790.

Hale, R., Fugate, D., Cetiner, M., Ball, S., Qualls, A., & Batteh, J. (2015). Update on ORNL TRANSFORM Tool: Preliminary Architecture/Modules for High-Temperature Gas-Cooled Reactor Concepts and Update on ALMR Control. *ORNL/SPR-2015/367, Oak Ridge National Laboratory*. URL: https://www.ornl.gov/publication/update-ornl-transform-tool-preliminary-architecture-modules-high-temperature-gas-0

Rader, J. D., Smith, M. B., Greenwood, M. S., & Harrison, T. (2019). "Nuclear Thermal Propulsion Dynamic Modeling with Modelica". URL: https://www.osti.gov/servlets/purl/1543223.

Yang, L., Liu, B., Shao, Y., Liang, T., & Tang, C. (2010). The failure mechanisms of HTR coated particle fuel and computer code. *Chinese Journal of Nuclear Science and Engineering*, *30*(3), 210-215, 222. URL: https://inis.iaea.org/search/search.aspx?orig_q=RN:45021255

Yanping, H., Xiaokang, Z., & Ji, D. (2021). Simulation Model Architecture and Concept Validation for Thermal Hydraulic Characteristics of 9.59.Two-Phase Fluid Based on Modelica[J]. *Nuclear Power Engineering, 42*(1), 1. DOI: https://doi.org/10.13832/j.jnpe.2021.01.0001

Energy-based Method to Simplify Complex Multi-Energy Modelica Models

Joy El Feghali[1] Guillaume Sandou[1] Hervé Guéguen[2] Pierre Haessig[2] Damien Faille[3]

[1]Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des signaux et systèmes, 91190, Gif-sur-Yvette, France, {joy.el-feghali,guillaume.sandou}@centralesupelec.fr
[2]IETR, CentraleSupélec, Rennes, France, {herve.gueguen,pierre.haessig}@centralesupelec.fr
[3]Electricité de France, R&D, PRISME Department, 78400, Chatou, France, damien.faille@edf.fr

Abstract

Energy production and consumption systems increasingly require more flexibility. The design of new control solutions can be a step, among others, towards flexibility. However, these control solutions often rely on the use of complex models, which are difficult to both manipulate and simulate. This paper presents a proof of concept of a method that reduces the complexity of multi-energy models modeled with Modelica language. This complexity-reducing method is based on simplifying the model's components that contribute less to the total energy using an energy-based ranking technique. The proposed solution is successfully applied to a complex city district model. A property of the Modelica language further allows redeclaration of low-ranked components without being compelled to fully redesign the model. Criteria verifying the multi-energy reduced model's precision, while respecting physical constraints, are also introduced.

Keywords: energy-based ranking, model reduction, multi-energy systems, Modelica

1 Introduction

A significant research effort is required in energy production to reduce fossil energy use and move towards a sizeable renewable energy penetration. Alongside this, optimizing energy consumption at the utility level (residential, tertiary, and industrial sectors) can increase global energy efficiency and decrease energy needs. The energy transition thus requires greater flexibility both on the production and consumption sides. This need has to be supported by new control systems. For this purpose, dynamical models are needed for control design.

The Modelica language (Fritzson and Engelson 1998) has been chosen as the modeling framework for dynamical models, thanks to its ability to capture multi-physics systems. However, this physical modeling may lead to complex or even intractable models, which cannot be used for control purposes and control law design. As a result, a suitable model obtained from a reduction of the full physical-based model is often necessary. This paper aims to provide a methodology to reduce the model's complexity by using a reduction technique applied to Modelica models.

In the model reduction literature, conventional methods are widely used, such as modal truncation method (Marshall 1966) and balanced truncation or Moore's method (Moore 1981). The modal truncation method aims to separate slow dynamics and fast dynamics in a modal base, while eliminating fast dynamics that influence the system less. With Moore's method, less controllable and less observable states are eliminated in a balanced base. These methods also require a linear system where eigenvalues and singular values are identified. However, physical models used for energy systems are not always linear, in which case energy models need to be linearized before applying reduction methods. This approach was used for Modelica models in Kim et al. (2014), building models were simplified in Modelica using physical properties and then reduced using Moore's method. Since this method is dedicated to LTI models, the physically simplified model of the building in Modelica was linearized before using the reduction method in Matlab. Although interesting for some specific components such as buildings, these methods appear hard to be generalized in the case of multi-energy systems due to the versatility of the non-linearities that need to be taken into account and the different operating conditions that should be considered.

Another drawback of these methods comes with the loss of the model structure. The models are indeed often obtained thanks to a change of basis in the model variables with difficult-to-interpret physical meaning. Methods, like the aggregation of states, allow preserving the structure of the model. Deng et al. (2014) applied this method to a non-linear building model. An analogy is used between the linear dynamics part and a continuous-time Markov chain to apply a Markov chain aggregation method. For Modelica models, additional work should be done to verify if the model is compatible with a particular state representation form before applying the method.

With the goal of maintaining the model's structure, Sodja, Škrjanc, and Zupančič (2019) and Sodja, Škrjanc, and Zupančič (2020) use an energy-based method applied to Modelica models, presenting reduction techniques for differential-algebraic equations (DAE) implemented in Modelica and reduction techniques for object diagrams.

DOI
10.3384/ecp21181577

The method is based on ranking procedures that help to eliminate equations or components with less of an influence over models. The methodology seems to be extendable to multi-energy systems and particularly well suited to the Modelica framework. Thus, this paper is based on the same approach used in Sodja, Škrjanc, and Zupančič (2019), where components are ranked according to their energy contribution to the system. The components that contribute less to the total energy of the system are simplified. This energy-based ranking method can be applied to Modelica language models after some adaptation to ease the energy flow calculation in the multi-energy system components. In this paper, the ranking-procedure method is applied on a multi-energy system benchmark provided by our industrial partner. The low-ranked components will be automatically replaced using a Modelica property. Criteria for local and global error calculation are introduced to maintain a good precision of the reduced model while respecting the physical constraints.

In Section 2, the energy-based ranking method used in Sodja, Škrjanc, and Zupančič (2019) will be detailed. In Section 3, a solution to reduce the model with simple user intervention is presented. Particularities of multi-energy models are highlighted, and the city district model used to test the simplification method is introduced in Section 4. In Section 5, the numerical results both from the simulation times and representativeness points of view are presented. In Section 6, additional perspectives are proposed.

2 Energy-based ranking method

The energy-based ranking method ranks the components of a model according to an energy metric. Initially introduced for bond graphs, this method was extended in Sodja, Škrjanc, and Zupančič (2019) to reduce Modelica models due to the similarity of the structure between the object-oriented models in Modelica and the bond graphs. While Modelica models can contain flow and effort variables, their product does not always correspond to the energy flow. Further modifications in the model are thus needed to calculate the energy flow. A metric is then chosen to rank the components of the model. Low-ranked components will be replaced with simplified component models.

2.1 Energy-flow calculation

All connections to a component i should be identified to calculate the energy flow in Modelica. The sum of all energy flows exchanged with other components k gives the total energy flow of a component i as defined in Equation 1.

$$\dot{E}_i(t) = -\sum_k \dot{E}_{i,k}(t) \tag{1}$$

The energy flow variable of each component defined in Equation 1 may not be directly available in the models and should be calculated by adding equations for energy flow calculation. Each Modelica component has a connector that links with a neighbor component k, and the information transferred through connectors helps to calculate the energy flowing to the component i. For example, with an electric pin connector of Modelica Standard Library, the voltage v and the current i are available. The product of these two variables gives the energy flow at the pin $\dot{E} = v * i$. Another example is a heat port connector of Modelica standard library; the temperature T and the heat flow rate Q_{flow} are available. The energy flow at the port is $\dot{E} = Q_{flow}$. Further examples of mechanics and fluid connectors can be found in Sodja, Škrjanc, and Zupančič (2019).

The energy flow is calculated during the simulation of the full model, after which the energy metric is calculated. For that purpose, a suitable energy metric should be chosen first.

2.2 Activity Calculation

Several metrics were introduced in (Sodja, Škrjanc, and Zupančič 2019), and the activity metric was chosen because it gives an idea of the error that the reduction causes. Initially, the activity was introduced in Louca (1998) for bond graphs. *Activity* of the component i, as defined in Equation 2, describes the energy flowing to a component i during a time interval $[t_1, t_2]$.

$$A_i = \int_{t_1}^{t_2} |\dot{E}_i(t)|\, dt \tag{2}$$

Each component that stores or dissipates energy will have an activity value; this helps to rank the components relative to their energy contribution to the system. The reduction idea is to simplify the components ranked at the bottom due to their low energy contribution. In the case of eliminating a component, its contribution is removed, causing a difference between the full and the reduced models. The activity ranking thus gives the error value from a global perspective.

Notice that the activity ranking is not absolute but depends on the simulation inputs and the time interval. Indeed, the activity integral in Equation 2 is computed on a given simulation duration with particular input signals. In the car suspension example in Sodja, Škrjanc, and Zupančič (2019), it is shown that a high-frequency road profile input (i.e., sharp edges) yields the elimination of slow-moving components and vice versa.

3 Modelica implementation of the replaceable components

A key question is how to remove a low-ranked component, or more precisely by which simplified model it should be replaced. The removal of a component leads to removing all of its connections. In Sodja, Škrjanc, and Zupančič (2019), the low-ranked components were removed manually by the user, and some modifications were needed to be done to avoid initialization problems. With large complex models, manual manipulation of the components may not be the best solution. Automatic modifications are

DOI 10.3384/ecp21181577

Listing 1. Example of the replaceable Envelope

```
model Building
replaceable OneZoneEnvelope Envelope
    constrainedby BaseEnvelope
annotation(choices(choice(redeclare
    OneZoneEnvelope Envelope), choice(
    redeclare EmptyEnvelope Envelope)));
    ...
```

thus needed. In this paper, the authors propose replacing the low-ranked component with a simplified compatible component model with a minimum user intervention. A simplified component model can be created with the same connections to other components as the full component model but without all of its equations that bring complexity to the model. These equations could be removed, linearized, or replaced depending on the physics of the component. Suppose low-ranked components are replaced with empty component models, where all internal connections and equations are removed. In that case, the energy-based component's contribution is set to zero or the maximum or the mean energy value. Several compatible models can be defined in order to replace low-ranked components. The replacement choice will be left to the user, depending on the study.

Modelica has properties that help to replace a component with another compatible component model without having to redesign the system. Suppose a model Building that contains an Envelope component and an EnergySystem component. At the upper-level of the model Building, the Envelope will be replaced with another simplified model by choosing from a list of predefined components.

The following Modelica keywords `replaceable`, `constrainedby`, `choices` and `redeclare` help replacing the components (Tiller 2014). When components are created in the full model, the keyword "replaceable" is added to the component declaration to address a property to the component that it can change its type.

The keyword "constrainedby" is used to specify a constraining type for all the new compatible types by which the component can be replaced. The constrained type can be a base component model that compatible models inherit from.

In order to create a list of component types, new type choices are added to the annotation of the replaceable component in the full model. The choices determine the component model types' possibilities for replacement. The example of the replaceable Envelope's declaration in the model is given in Listing 1. At the upper-level of the complete model, using the keyword "redeclare", the replaceable component changes its type. The user can choose from a list of predefined component models and replace the original component model. The expression with "redeclare" is automatically added to the declaration of the component in the Modelica model, at the upper-level, when choosing from a list. An example of the text in Modelica for the Envelope redeclare at the upper level is shown in Listing 2.

Listing 2. Example with redeclare at the upper-level of the model Building

```
model BuildingUpperLevel
Building bldg(redeclare EmptyEnvelope
    Envelope);
end BuildingUpperLevel;
```

4 Multi-energy system case-study

In this paper, the interest of the authors is to reduce the complexity of multi-energy models in Modelica by applying the energy-based reduction method. Multi-energy districts are considered, combining different types of energy production and consumption. These multi-energy districts are formed of buildings interacting with an electrical grid and a district heating network. The two latter systems can interact by a combined heat and power unit or by the electrical consumption of the district heating network. Renewable energy systems can also be connected.

The energy-based reduction method can be applied to these district Modelica models by choosing buildings as components to be ranked. Since buildings interact with different types of energy systems, they will have multiple energy flow variables. The energy flowing to a building is the power demanded from each of the connected energy systems. With a district composed of buildings communicating their power demands while interacting with an electrical grid and a district heating network, each building will have three types of activities: the active electrical activity $A_{P_{elec}}$, the reactive electrical activity $A_{Q_{elec}}$ and the heating activity $A_{P_{heat}}$. These variables correspond to the different energies flowing to the building for a chosen interval. Due to the multi-energy aspect of the system, these variables are separated.

After applying the energy-based ranking, the low-ranked buildings will be replaced with a simplified buildings model. At the source level, the error between the full and reduced models can be determined with the activity ranking. However, due to the interaction of the buildings with other systems like the electrical grid and heating network, removing a building causes error from a local perspective in the electrical grid model and heating network model. The precision of the reduced model should be checked at all levels, and the physical constraints of the model should be respected. For example, physical constraints for the electrical grid are checked for the voltage and the current values of the lines. The voltage should stay within a range of $\pm 5\%$ of the nominal medium voltage for medium voltage lines and $\pm 10\%$ of the nominal low voltage for low voltage lines. The current value should not exceed the I_{max} of the line.

Criteria of signals in the electrical grid and district heating network models are used to validate the reduced model depending on the system's physics. Criteria help to analyze how the signal is affected by the removal of the component.

The principles of the reduction of multi-energy models are applied in the case of a city district provided by Électricité de France (EDF R&D). The system is modeled using EDF R&D Modelica libraries.

4.1 Description of the model

The chosen system is a sizeable multi-energy district located in the southern suburbs of Paris described by the *PowerGrid* demonstrator (Bouquerel et al. 2019). The district is composed of 719 buildings, one electrical grid, and one district heating network. The district's model in Dymola is complex due to its scale, and the computation time is thus long, raising the need for a reduction method. A smaller use case of 20 buildings is issued from the PowerGrid model to test the energy-based reduction method before applying it to a larger-scale model. These 20 buildings are connected to an electrical grid, and 12 of them are also connected to a district heating network. The buildings are local producers; they produce energy through photovoltaic panels (PV panels). The district heating network comprises a heat pump assisted by a storage tank that delivers hot water through pipes. EDF R&D Modelica libraries are used to model the system in Dymola: BuildSysPro (Plessis, Kaemmerlen, and Lindsay 2014) for buildings, PowerSysPro (Tavella 2020) for the electrical grid, and MixSysPro, an internal EDF R&D library for the district heating network.

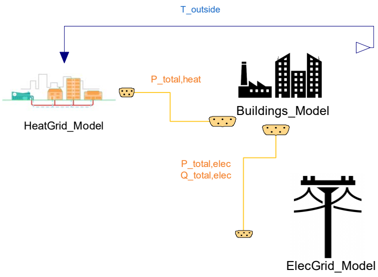


Figure 1. The multi-energy model in Dymola

The upper-level of the district model is composed of the buildings model, the electrical grid model, and the heating network model as shown in Figure 1.

In the buildings model as shown in Figure 2, the system is composed of an envelope model and an energy-system model. The envelope model calculates the interior temperature that is used in the energy-system model. In the energy-system model, the electrical power and heating power demands are calculated according to the consumption scenario. Buildings have solar panels, so they produce electrical power that the building will consume. If the PV production is higher than the need, the rest will be delivered to the electrical grid. The outputs of the buildings' model are the power demanded from the heating network and electrical grid. Each building connected to the electrical grid share its active $P_{total,elec}(t)$ and reactive electrical power $Q_{total,elec}(t)$. These electrical powers add all the electrical consumption demands of a building while reducing the amount produced locally by the PV panels. Each building connected to the heating network share its total heating consumption $P_{total,heat}(t)$. The heating power includes the hot water demand and the heater demand.

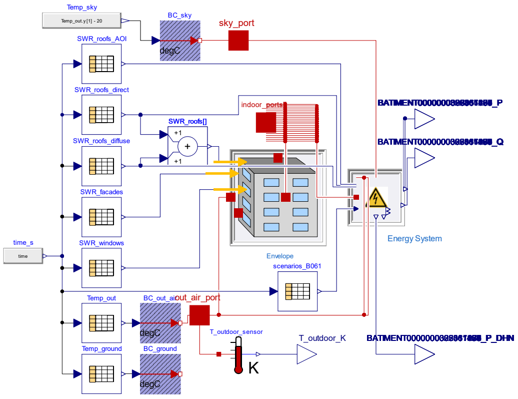


Figure 2. Buildings model in Dymola

The electrical grid model is composed of 23 lines. The 20 buildings are connected to low voltage or medium voltage lines as shown in Figure 3. The inputs of the electrical grid model are the electrical power demands of each building. The current and voltage of the lines are calculated within the model with power flow calculation.

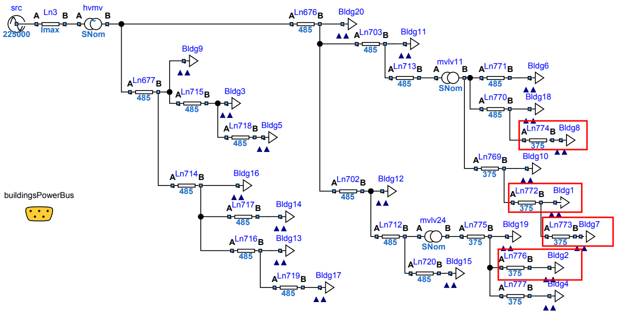


Figure 3. Electrical grid model in Dymola. The highlighted framed buildings are the subject of the subsection 5.2

The heating network model in Figure 4 contains the heat pump and the storage tank that feed the 12 substations connected to the 12 buildings. The buildings are assigned to one of the three subnetworks. The red and blue connections correspond to the hot water and cold water circulations, respectively. The inputs of the heating network model are the heating power demands of each building. The substations' return temperature $T_{return}(t)$ is calculated in the heating network model and depends only on the outside temperature. All buildings thus have the same return temperature $T_{return}(t)$. The mass flow is calculated for a building i connected to a substation i: $\dot{m}_i(t) = \frac{P_{total,heat_i}(t)}{c_p(T_{supply_i}(t) - T_{return}(t))}$ with $T_{supply_i}(t)$ the temperature received at the substation i and c_p the fluid specific heat capacity.

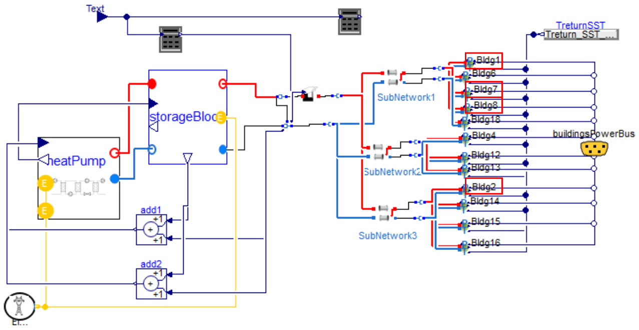


Figure 4. District heating network model in Dymola. The highlighted framed buildings are the subject of the subsection 5.2

4.2 Impact of the reduction on the multi-energy model

The simplification of the low-energy buildings affects both the electrical grid and district heating network models from global and local perspectives.

For the electrical grid, the global level corresponds to the electrical source level. The local level corresponds to the medium voltage and low voltage lines. Power, voltage, and current values of the lines will change in the electrical grid model due to the building simplification. The electrical power at the source level will vary slightly when removing a building with low energy contribution. However, at the line directly connected to the building removed, there will be a 100% error of the power, but which is supposed to be non-significant for the rest of the grid. The power error can be deduced from the activity ranking and the lines' position according to the buildings removed. The error of the current and voltage of the lines should be calculated by simulating the reduced model and comparing values with the full model. The voltage at a line is not allowed to vary significantly compared to the nominal value; the threshold is chosen to be 1%. The reduced model can be validated if the voltage and current values at the electric lines do not exceed the physical allowed limits.

For the district heating network, the global level corresponds to the heating source, and the local level is at the substations. At the heat pump and storage tank level, the power error will be minor when a low-energy building is removed. At the substation's level, there will be a 100% error of power. In the district heating network, the temperature and the mass flow should be analyzed. The reduced model's mass flow and temperature should not vary significantly compared to the full model's values. At the substation level of the building removed, the mass flow will be equal to zero because the power is set to zero. The substation's model ensures a minimum mass flow. The temperature $T_{supply_i}(t)$ received by a building i may vary when removing a building on the same subnetwork due to losses in the pipes that depend on the mass flow.

4.3 Replacing buildings with empty buildings models

As stated in section 3, a low-ranked component could be replaced with different types of component models; different possibilities thus exist for building replacement. When the building's energy contributions are chosen to be eliminated, as done in this study, the active and reactive electrical power values and the heating power values are set to zero. Later on, it can be set to another value. Since the building model comprises an envelope model and an energy-system model, a predefined empty model is created for each.

All subcomponents and internal connections are removed from the envelope and energy-system models; connectors and parameters only remain. The removal of the internal connections decreases the number of equations of the full model. The energy system model's outputs are the active and reactive electrical power values connected to the electrical grid and the heating power value connected to the district heating network. These values are set to zero.

Other surrogate models of buildings can be defined, like an energy-system connected only to the electrical grid or the heating network. Energy systems without PV panels can also be possible. It would be interesting to compare all the results of these possibilities of simplification. This paper's work is limited to replacing the envelope model and the energy-system model with empty models.

5 Results

The energy-based ranking method is applied to the district model. The full model does not have to be simulated to obtain the ranking; the simulation of the buildings model is only needed. In the case where the model of the buildings is too large for the solver and cannot be simulated, each building can be simulated separately. In other applications, a limitation of the method can appear when the model cannot be simulated to calculate the activities of the components.

For each building, three types of activities are analyzed: the active electrical activity $A_{P_{total,elec}}$, the reactive electrical activity $A_{Q_{total,elec}}$ and the heating activity $A_{P_{total,heat}}$. The three activities are obtained by applying the Equation 2 for a one-year interval with $P_{total,elec(t)}$, $Q_{total,elec}(t)$ and $P_{total,heat}(t)$ considered as the three energy flowing to the building. The horizon could also be chosen to be a single season of interest. However, in this case study, all buildings share the same synthetic consumption scenario.

Thus the effect of seasonal variations affects activity magnitudes but cancels out in the activity ranking.

The activities are calculated for each of the 20 buildings during the simulation. The ranking of the buildings for each of the three activities is then deduced. The ranking depends on the model, especially on the input scenario and the parameters of the system. As the ranking is roughly the same between the three tables' shared buildings, only the Table 1 from the electrical active power point of view is presented. The buildings that will be simplified are easily identified.

Differences in the ranking of the three activities can happen due to a difference in the scenario between buildings or a change in the time slot. In this case, a suitable replacement building model will have to be found that combines the simplification according to multiple ranking.

A limitation of this method is when a majority of buildings have similar and low activity values. Then, it is not possible to remove low activity buildings while preserving most of the total activity of the system. Aggregation might be an alternative solution to combine buildings with similar properties. When using aggregation, the nodes of each of the district heating network and the electrical grid should be aggregated as well. A work on aggregating the nodes of a Modelica district heating network model can be found in Falay et al. (2020).

Table 1. Ranking of the buildings from the electrical active power point of view

Buildings	*Activity*	*Relative*	*Accumulated*
	[MWh]	*(%)*	*(%)*
Building 20	2 270	26	26
Building 15	1 294	15	41
Building 16	890	10	51
Building 14	627	7	58
Building 12	618	7	65
Building 17	612	7	72
Building 13	526	6	78
Building 03	400	5	82
Building 06	325	4	86
Building 11	260	3	89
Building 18	223	3	91
Building 05	197	2	94
Building 09	151	2	95
Building 10	143	2	97
Building 04	95	1	98
Building 19	77	1	99
Building 08	51	1	100
Building 07	14	0	100
Building 02	14	0	100
Building 01	13	0	100
Total	*8 800*		

5.1 Simulation time

The multi-energy model, shown in Figure 1, is simulated for a one-year period with one-hour sampled inputs using CVODE solver with a variable step. The building model interacts with the electrical grid and district heating network models; this interaction causes higher simulation time. The simulation time is expected to decrease when buildings are simplified. The number of equations decreases linearly as in Figure 5 with the decrease of the number of buildings to simulate. The simulation time does not follow a linear decrease as shown in Figure 6. For example, simplifying the four low-ranked buildings leaves 16 buildings to simulate. The full model's simulation time is thus reduced by almost a factor of two.

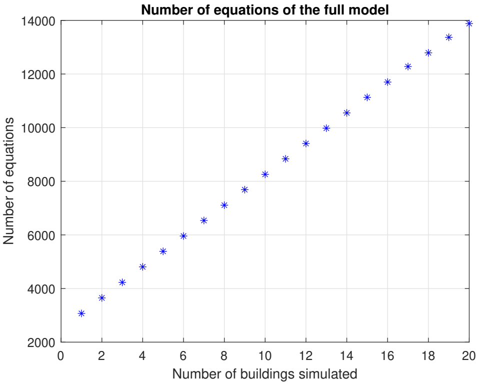


Figure 5. Number of equations of the full model

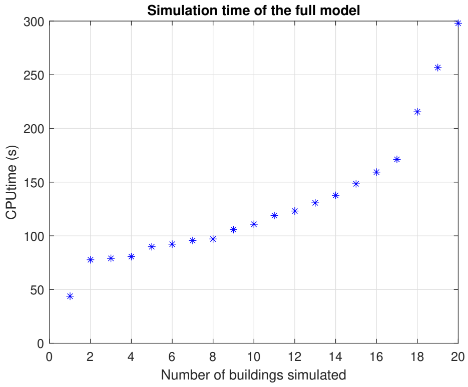


Figure 6. Simulation time of the full model

5.2 Reduced model verification

After the reduction is applied, the accuracy of the city district reduced model should be verified. Criteria for several signals are defined to calculate the error generated by the simplification of the buildings. The *red* indices in the criteria definitions correspond to the values of the signals obtained with the reduced model's simulation.

Criteria are introduced for the electrical grid verification in Equation 3, Equation 4 and Equation 5 with $I(t)$, $U(t)$ the values of the current and voltage at a line obtained with the full model's simulation. U_{nom} is the nominal value of the voltage at a line. Equation 4 represents the mean absolute error (MAE) of the current relative to the current's maximum value. T is the time span of the signal values. Equation 6 and Equation 7 are introduced for the heating network with $T_{supply}(t)$, $\dot{m}(t)$ the values of the received temperature and the mass flow at a substation of a building obtained with the full model's simulation.

$$CI_{\infty}(line\ i) = \frac{||I(t) - I_{red}(t)||_{\infty}}{||I(t)||_{\infty}} \tag{3}$$

$$CI_{MAE}(line\ i) = \frac{\frac{1}{T}\int_0^T |I(t) - I_{red}(t)|}{||I(t)||_{\infty}} \tag{4}$$

$$CU_{\infty}(line\ i) = \frac{||U(t) - U_{red}(t)||_{\infty}}{U_{nom}} \tag{5}$$

$$CT_{\infty}(building\ i) = \frac{||T_{supply}(t) - T_{supply,red}(t)||_{\infty}}{||T_{supply}(t)||_{\infty}} \tag{6}$$

$$C\dot{m}_{\infty}(building\ i) = \frac{||\dot{m}(t) - \dot{m}_{red}(t)||_{\infty}}{||\dot{m}(t)||_{\infty}} \tag{7}$$

The case study is composed of 20 buildings. The four buildings ranked last in the ranking of Table 1 are removed at once, which leaves a model of 16 buildings to simulate. The removal of the contribution of these buildings affects the variables' values of the electrical grid and heating network, which should be validated using the defined criteria. The electrical grid is composed of 23 lines connected to medium and low voltage buildings. After reducing the buildings, the current of these lines does not surpass the maximum value I_{max} allowed of the line, and the voltages at the lines are far from the limits. Equation 3, Equation 4, and Equation 5 are represented in Figure 7 and Figure 8.

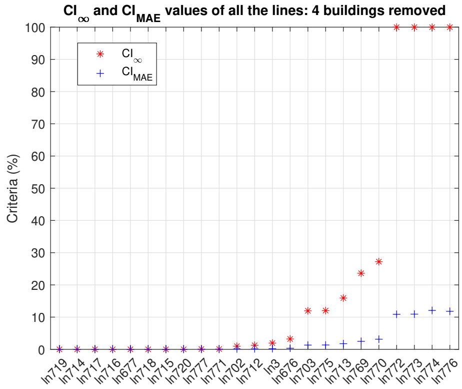


Figure 7. Error on the current of the lines of the electrical grid with respect to Equation 3 and Equation 4

In Figure 7, the error is 100% for the lines 772, 773, 774 et 776 because these lines are directly connected to the four removed buildings (see Figure 3). For the other lines, the error on the current does not exceed 30% of the maximum value of the current with the full model's simulation, and the MAE criterion values are small. These values are acceptable since the current values of the reduced model are

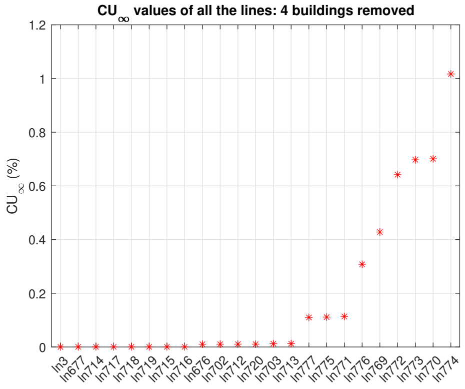


Figure 8. Error on the voltage of the lines of the electrical grid with respect to Equation 5

far from the I_{max} limits. In Figure 8, the criteria on the voltage values do not exceed 1% of the nominal voltage value of the line, except for line 774, where the building is removed. Voltages are less sensitive than the currents to the variation of the power values of the buildings. The error on the lines depends on the position of the line relative to the building removed. In Figure 9, the different criteria values are presented for line 3 at the source level with respect to the number of buildings simulated. As expected, the current criteria values increase when more buildings are removed. The voltage at the line 3 does not vary significantly.

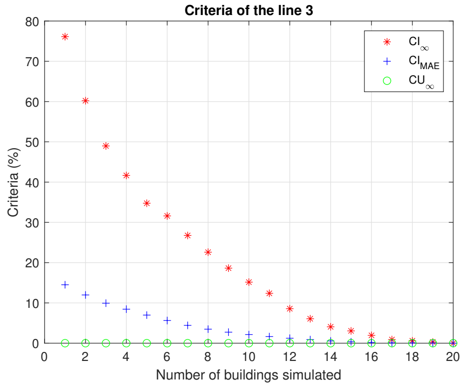


Figure 9. Criteria of the line 3 while removing buildings

The district heating network is affected when removing buildings. The power demand is set to zero; this sets the building's mass flow $\dot{m}_{red}$ of the reduced model to zero. Building 1, 2, 7, and 8 are the four removed buildings, and they have an error of mass flow of 100% between the full and reduced model, not shown in Figure 10. The other buildings are negligibly affected depending on their positions relative to the removed buildings (see Figure 4). In Figure 11, buildings on the same subnetwork have the same temperature error; this is because of the way of modeling the losses in the pipes in the district heating network. Buildings on subnetworks that are not affected have a zero

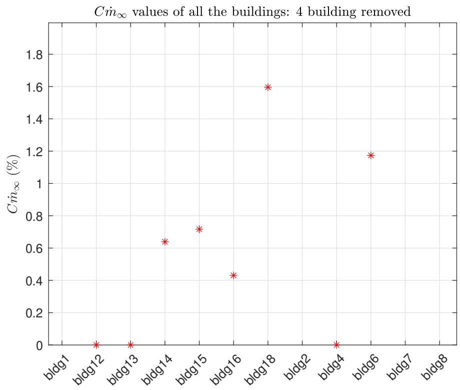


Figure 10. Error on the demanded mass flow of the buildings of the district heating network with respect to Equation 7

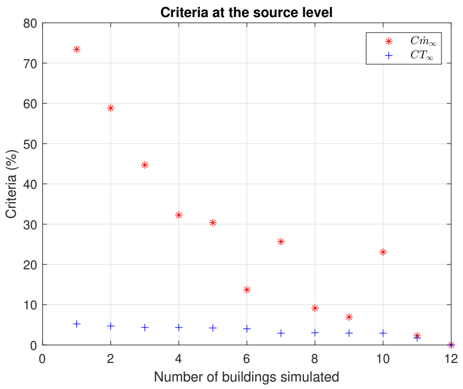


Figure 12. Criteria at the source level of the district heating network while removing buildings

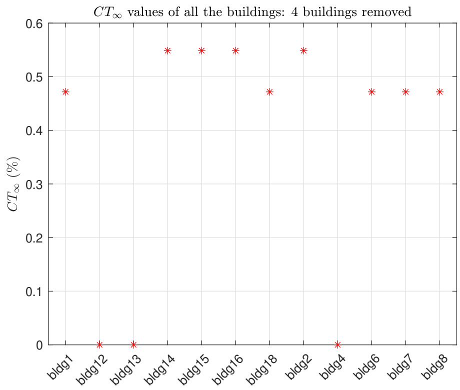


Figure 11. Error on the received temperature of the buildings of the district heating network with respect to Equation 6

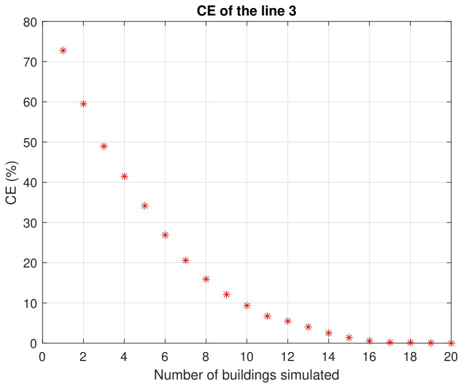


Figure 13. Error on the energy at the electric source level

error. In Figure 12, the different criteria values are presented at the heating source level for the number of buildings simulated. The error is expected to increase when buildings are simplified. However, mass flow error values are high when simulating 5, 7, and 8 buildings. No explanation is yet certain, but this result is affected by the subnetwork assigned to the removed buildings, where there are heat losses.

An energy criterion at line 3 is defined in Equation 8, where $\sum_k |P_{total,elec}(k)|$ is the sum of one-hour samples over a one year simulation (8761 samples). The results in Figure 13 reflect the error on the demanded power, caused by the removed building's contribution from Table 1.

$$CE(line\ 3) = \frac{|\sum_k |P_{total,elec}(k)| - \sum_k |P_{total,elec,red}(k)||}{\sum_k |P_{total,elec}(k)|} \tag{8}$$

6 Conclusion

In this paper, a proof of concept for the method of reducing complex multi-energy models is presented. This method conserves the model's physical meaning by reducing the model's components using energy-based ranking. Components are removed or replaced with a simpler model using the replaceable and redeclare properties. The reduced model has a shorter simulation time, the precision is evaluated by the criteria proposed, and the physical constraints were respected. It is interesting for future work to find a suitable number for removed buildings by compromising between the simulation time and the model's precision. An estimation of the local error at the electrical grid and heating network levels is practical and replaces simulating the reduced model and the full model to compare them. As a next step, we will test our method on a larger scale model and on a different time horizon and inputs. For cases when this ranking-based model reduction doesn't apply well (e.g. when a large fraction of components share similar low activity values), a to-be-defined component aggregation strategy would be complementary.

Acknowledgements

This work has been supported by Électricité de France (EDF R&D) and RISEGrid Institute. This work reuses parts of the PowerGrid demonstrator which was partly funded and supported by BPI France through the French FUI ModeliScale research project. The authors would like to thank Jean-Philippe Tavella, Dominique Croteau, Hassan Bouia, and Clément Flinois for their support with the model implementation and expertise in the various fields.

DOI
10.3384/ecp21181577

References

Bouquerel, Mathias et al. (2019). “Requirements modelling to help decision makers to efficiently renovate energy systems of urban districts”. In: *Proceedings of the 2019 Summer Simulation Conference*, pp. 1–12.

Deng, Kun et al. (2014). “Structure-preserving model reduction of nonlinear building thermal models”. In: *Automatica* 50.4, pp. 1188–1195.

Falay, Basak et al. (2020). “Enabling large-scale dynamic simulations and reducing model complexity of district heating and cooling systems by aggregation”. In: *Energy* 209, p. 118410.

Fritzson, Peter and Vadim Engelson (1998). “Modelica—A unified object-oriented language for system modeling and simulation”. In: *European Conference on Object-Oriented Programming*. Springer, pp. 67–90.

Kim, Eui-Jong et al. (2014). “Urban energy simulation: Simplification and reduction of building envelope models”. In: *Energy and Buildings* 84, pp. 193–202.

Louca, Loucas Sotiri (1998). “An energy-based model reduction methodology for automated modeling.” PhD thesis.

Marshall, SA (1966). “An approximate method for reducing the order of a linear system”. In: *Control* 10, pp. 642–653.

Moore, Bruce (1981). “Principal component analysis in linear systems: Controllability, observability, and model reduction”. In: *IEEE transactions on automatic control* 26.1, pp. 17–32.

Plessis, Gilles, Aurelie Kaemmerlen, and Amy Lindsay (2014). “BuildSysPro: a Modelica library for modelling buildings and energy systems”. In: *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*. 096. Linköping University Electronic Press, pp. 1161–1169.

Sodja, Anton, Igor Škrjanc, and Borut Zupančič (2019). “Realization-preserving model reduction of object-oriented models using energy-based metrics”. In: *Simulation* 95.7, pp. 607–620.

Sodja, Anton, Igor Škrjanc, and Borut Zupančič (2020). “Cyber-physical modelling in Modelica with model-reduction techniques”. In: *Journal of Systems and Software* 163, p. 110517.

Tavella, Jean-Philippe (2020). *PowerSysPro library*. URL: https://bitbucket.org/simulage/powersyspro/wiki.

Tiller, Michael (2014). *Modelica by Example*. URL: https://mbe.modelica.university/components/architectures/replaceable/.

A Case Study on Condenser Water Supply Temperature Optimization with a District Cooling Plant

Kathryn Hinkelman[1] Jing Wang[1,2] Chengliang Fan[1,3] Wangda Zuo[1,2] Antoine Gautier[4]
Michael Wetter[4] Nicholas Long[2]

[1]Department of Civil, Environmental and Architectural Engineering, University of Colorado Boulder, USA,
{kathryn.hinkelman,jing.wang,chengliang.fan,wangda.zuo}@colorado.edu
[2]National Renewable Energy Laboratory, USA, nicholas.long@nrel.gov
[3]School of Civil Engineering, Guangzhou University, China
[4]Lawrence Berkeley National Laboratory, USA, {agautier,mwetter}@lbl.gov

Abstract

District cooling (DC) continues to proliferate due to increasing global cooling demands and economies of scale benefits; however, most district-scale modeling has focused on heating, and to the best of our knowledge, researchers have yet to model cooling plants featuring waterside economizers in DC settings. With the Modelica Buildings library expanding its capabilities to district scale, this study is one of the first to demonstrate how the open-source models can be used for detailed energy and control analysis of a DC plant. For a real-world case study, we developed and validated high-fidelity models for a DC central plant at a college campus in Colorado, USA, and we optimized the condenser water supply temperature (CWST) setpoint across multiple time horizons using the Optimization library in Dymola. Results indicate that annual CWST optimization saves 4.7% annual plant energy, with less than 1% of additional energy savings gained through daily optimization. This confirms previous studies' findings that high frequency CWST optimizations are not necessary for the studied system.

Keywords: District Cooling, Optimization, Chiller Plant, Waterside Economizer, Modelica Buildings Library

1 Introduction

District cooling (DC) systems typically provide cooling services to buildings from central plants and are increasing in demand. In the United States for example, DC serves 174 million square meters of floor space, delivering 15 GW of chilled water annually (ICF LLC and International District Energy Association 2018). This is currently more than any other country, but global installations are growing rapidly, particularly in the Middle East (Marafeq Qatar 2015). With buildings consuming 36% of global energy (International Energy Agency 2019) and space cooling growing faster than any other end use (International Energy Agency 2018), many are looking to DC for its energy efficiency and economic benefits (Anderson, Rezaie, and Rosen 2021; Oppelt et al. 2016; Zabala et al. 2020). Rather than individual buildings producing their cooling needs with individual air conditioning equipment, centralized plants produce chilled water (CHW) that can be distributed to multiple buildings connected to the district. This aggregation of cooling equipment to a district scale enables the centralized maintenance, the use of more efficient chillers, and the integration of renewable energy resources.

Current modeling and simulation work tends to focus on district heating (DH) with limited focus on DC. A simple Scopus search involving the keywords “model” and either “district heating” or “district cooling” produces 20,109 and 1,230 results, respectively. While some DH research can be applied to DC – as suggested in some DH case studies (del Hoyo Arce et al. 2018; Falay et al. 2020; van der Heijde et al. 2017) – there are also important differences that make DC modeling unique. For example, cooling generation efficiency has heightened sensitivity to even small changes in CHW temperature (e.g., $0.1K$) (Oppelt et al. 2016), and "low delta-T syndrome" (ASHRAE 2013) is a common energy efficiency problem among DC systems and chiller plants.

Several groups have made valuable contributions to DC modeling literature. High-fidelity and reduced-order modeling techniques have been adapted to reduce plant energy consumption (Chow et al. 2004), peak loads (Gang et al. 2015), and implement model predictive control (Zabala et al. 2020; Matsouka and Hill 2020), to name a few. While a variety of chiller types have been studied – including compressor, absorption, turbo, and double-effect varieties – to the authors' best knowledge, none of the previous literature modeled chiller plants with waterside economizers (WSEs) in DC applications. Further, we only found one study that used Modelica for DC plant modeling (Zabala et al. 2020); yet Modelica is a promising platform for these applications due to its acausal modeling scheme, multitude of variable time-step numerical solvers, and rich open-source libraries with high re-usability potential. This work demonstrates how the popular, open-source Modelica Buildings library can be applied for detailed modeling of chiller plants with WSE for DC applications.

The university wants to identify energy efficiency im-

DOI
10.3384/ecp21181587

provements with little to no financial investments in equipment upgrade. Thus, condenser water supply temperature (CWST) optimization was selected for its past successes in reducing chiller plant energy consumption (Lan Wang, Lee, and Yuen 2018). The condenser water supply is the water entering the condenser of the chillers, and its temperature setpoint affects the chillers' operating efficiency, the economizing heat exchanger's effectiveness, and the required cooling tower fan power. Several past works in chiller plant simulation include CWST optimization (Karami and Liping Wang 2018; Ling et al. 2018), and several optimization time horizons from hourly to monthly have been studied (Huang, Zuo, and Sohn 2017).

In this work, we modeled the DC plant for an existing college campus featuring six connected buildings in Colorado, United States (ASHRAE Climate Zone 5B). The objectives of this case study are to (1) demonstrate the application of Modelica and the Buildings library for detailed energy analysis of a DC plant with a WSE, and (2) identify the the optimal CWST setpoint by evaluating several optimization time horizons. While we selected CWST optimization for this case study, it is important to note that the model can be used for other analyses as well, such as replacing the chillers or adding thermal storage. In Section 2, we present the mechanical and control systems for the case study DC plant. This is followed by the Modelica implementation in Section 3, and the verification and validation of equipment and system models in Section 4. Presentation of the optimization methodology and the optimization results are in Sections 5 and 6. Section 7 concludes the paper with future work.

2 System Description

The case study site is a college campus in Colorado's Denver Metropolitan area with a central plant providing chilled water for space and process cooling to six buildings. This section presents the mechanical and control systems for the DC plant.

2.1 Mechanical System

As depicted in Figure 1, the cooling plant is a primary-only chilled water system with parallel connections between a WSE and two chillers on both the plant side (the condenser water (CW) piping) and the load side (the CHW piping). Following standard nomenclature, the condenser water supply (CWS) is the plant-side water being supplied to the chillers, and the return (CWR) is returning to the cooling towers. Similarly, the chilled water supply (CHWS) is being supplied to the district, while the return (CHWR) is returning to the plant. Both the CW and CHW loops contain bypasses. The CW bypass valve is a two-position directional valve to switch between cooling tower and bypass modes, while the CHW bypass valve modulates to maintain the minimum CHW flow rate through the evaporator of the chillers. Although both the CW and CHW pumps are equipped with variable frequency drive (VFD) motor controllers, the CW pumps modulate their speed to maintain a constant flow rate setpoint, while the primary-only CHW pumps operate at variable speeds to maintain a differential pressure setpoint at a distant building. Further details regarding the nominal equipment information can be found in the Appendix (Table 3).

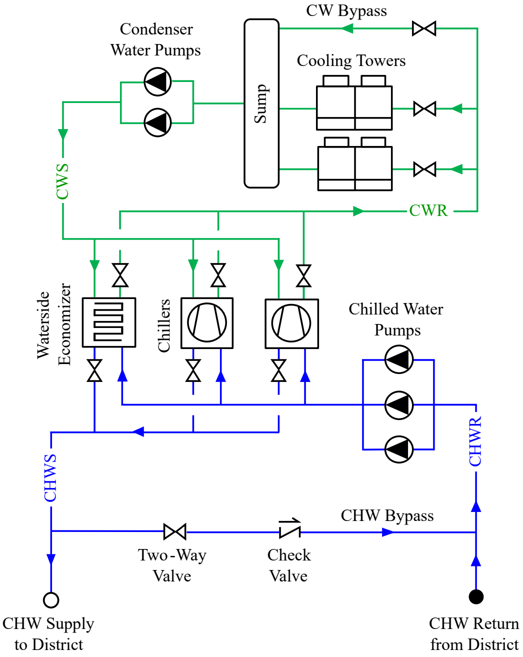


Figure 1. Schematic diagram for the central plant.

2.2 Control System

The control system includes four levels (Figure 2): a top-level Master Control, Systems Control, Units, and Devices. First, the *Master Control* determines the operating state of the entire plant and sequences the various "systems". Second, the *Systems Control* represents the collection of similar "units" physically connected in the process loop. This control level determines correct number of units that should be running to meet the demand (e.g., staging of various equipment). Third, *Units* represent the collection of devices that combine to perform a specific task. The Units Control level prescribes the setpoint for equipment operation. Lastly, the *Devices* layer contains single-input single-output (SISO) systems, providing the fundamental building blocks of the control. These are local control setpoints predominantly met by proportional integral (PI) controllers.

At the top Master Control level, the cooling plant can operate in three active cooling modes in addition to the *Off* mode: (1) *Free Cooling* (FC) mode, (2) *Mechanical Cooling* (MC) mode, and (3) *Pre-Mechanical Cooling* (Pre-MC) mode. The state graph in Figure 3 depicts the switching conditions to move between each of these states. Switching conditions include the total cooling load $\dot{Q}_C$ (calculated from temperature and mass flow sensors at the plant); the wetbulb temperature WBT and its switching setpoint WBT_{Set}; the chilled water mass flow rate $\dot{m}_{CHW}$;

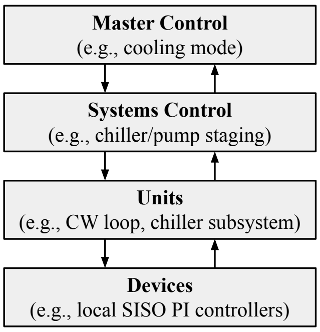

Master Control
(e.g., cooling mode)
Systems Control
(e.g., chiller/pump staging)
Units
(e.g., CW loop, chiller subsystem)
Devices
(e.g., local SISO PI controllers)

Figure 2. Four control levels of the central plant.

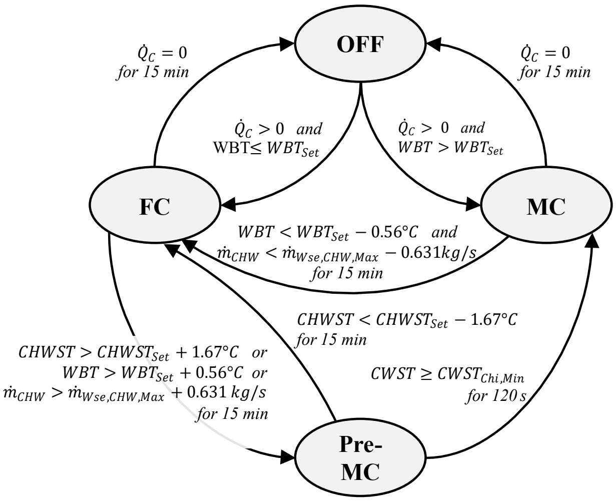

OFF
FC
MC
Pre-MC
$\dot{Q}_C = 0$ for 15 min
$\dot{Q}_C = 0$ for 15 min
$\dot{Q}_C > 0$ and WBT≤ WBT_{Set}
$\dot{Q}_C > 0$ and $WBT > WBT_{Set}$
$WBT < WBT_{Set} - 0.56°C$ and $\dot{m}_{CHW} < \dot{m}_{Wse,CHW,Max} - 0.631 kg/s$ for 15 min
$CHWST < CHWST_{Set} - 1.67°C$ for 15 min
$CHWST > CHWST_{Set} + 1.67°C$ or $WBT > WBT_{Set} + 0.56°C$ or $\dot{m}_{CHW} > \dot{m}_{Wse,CHW,Max} + 0.631\ kg/s$ for 15 min
$CWST \geq CWST_{Chi,Min}$ for 120 s

Figure 3. Master control logic for selecting plant cooling mode.

the maximum allowable chilled water mass flow rate for the WSE $\dot{m}_{Wse,CHW,Max}$; the chilled water supply temperature $CHWST$ and its setpoint $CHWST_{Set}$; the condenser water supply temperature $CWST$; and the minimum condenser water supply temperature allowed by the chiller $CWST_{Chi,Min}$. The offset and dead band temperature of the control signals as well as the waiting times are adjustable. For this plant, the maximum WSE chilled water mass flow rate is 120.5 kg/s. The wetbulb temperature transition setpoint, the chilled water supply temperature setpoint, and minimum chiller condenser water supply temperature are 6.7°C, 6.1°C, and 10.0°C, respectively.

3 Modelica Implementation

The DC plant is implemented in Modelica using components from the Modelica Buildings library version 7.0.0 (Wetter et al. 2014) and Modelica Standard Library version 3.2.3. New system and equipment-level models were developed as part of this study, which will be open-source released in the Modelica Buildings library. The system models are presented in a top-down approach in the following sections.

3.1 Mechanical System

Shown in Figure 4, the central cooling plant model contains several control blocks on the left with the condenser water (green lines) and chilled water (blue lines) loops on the right. The system's design schematic (Figure 1) and Modelica diagram contain one-to-one modeling relationships, allowing users to clearly interpret the configuration. We connected the inlet and outlet ports of the plant to a district model that reflected the tabulated heat flow rate (broken down by mass flow rate and change in CHW temperature) of the real district from 2018 measured data.

For this case study, the cooling plant features two chillers with a WSE connected in parallel on both the chilled water and condenser water sides. We instantiated the *Buildings.Applications.DataCenters.ChillerCooled. Equipment.Nonintegrated* model with the optional CHW supply temperature control on the WSE disabled, which implements the *ElectricEIR* chiller model, based on the DOE-2 electric chiller (Hydeman and Zhou 2007).

New subsystem models for the cooling tower with CW bypass and a parallel cooling tower model were developed based on the Modelica Buildings and Modelica Standard libraries. For the cooling tower model, we instantiated the Merkel model from the Modelica Buildings library, based on the variable speed Merkel model in EnergyPlus version 8.9.0 (United States Department of Energy 2018).

The chilled water pump subsystem was modeled as three parallel speed controlled pumps with inline isolation valves. For the constant speed condenser water pumps, the subsystem included two parallel mass flow controlled pumps. When appropriate, flow controlled pumps (as opposed to speed controlled pumps) typically reduce the size of the nonlinear system of equations in the model, which in turn reduces the simulation run time. However, to note, modelers should use caution when evaluating the energy consumption of ideal pumps that enforce the flow rate regardless of head, because if the pump works against a closed valve, then unrealistic electric power spikes can occur because the power is proportional to the product of the enforced mass flow rate times the pump head, which can be arbitrarily high for this idealized model.

3.2 Control System

The four control layers are implemented in Modelica. The *Master* control (Figure 5) mirrors the schematic state diagram shown previously in Figure 3. Six real inputs decide the state of the *Master* control mode: Off, FC, MC, or Pre-MC. An integer output ranging from 0 to 3 corresponds to the cooling mode status. This control is packaged as one block and instantiated in the top-level system model for the central plant. All *Systems* control blocks follow a similar implementation.

Figure 6 exemplifies the CW loop control implementation. This includes determining the operating state through the CW control mode staging (*Systems* level), specifying the temperature setpoint in the CW loop subsystem (*Units* level), and implementing the local PI controllers for the cooling tower fan and bypass valves (*Devices* level). Depending on the cooling mode (FC, MC, Pre-MC), either the chilled water or condenser water sup-

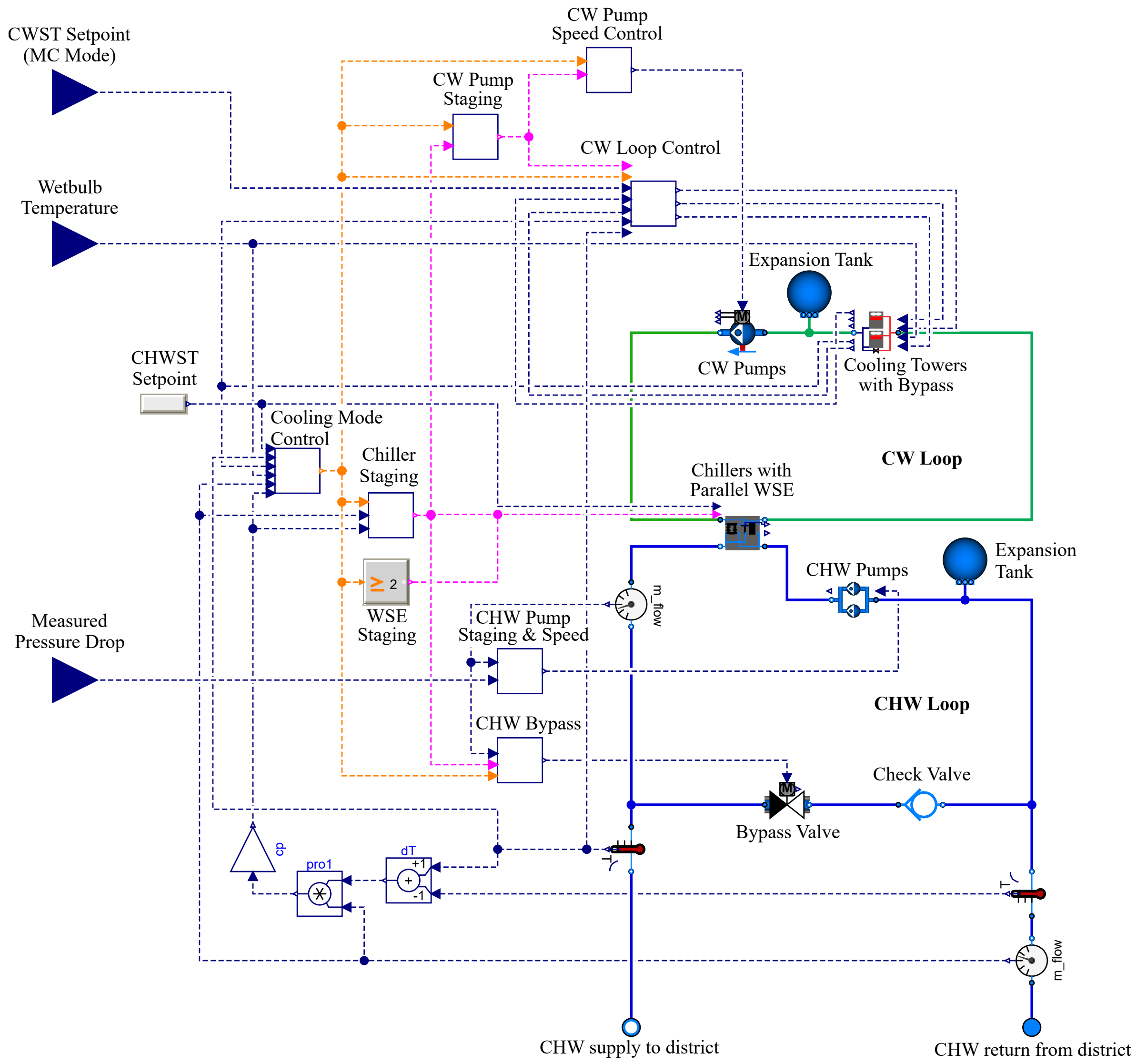


Figure 4. Diagram of Modelica model for the district cooling plant.

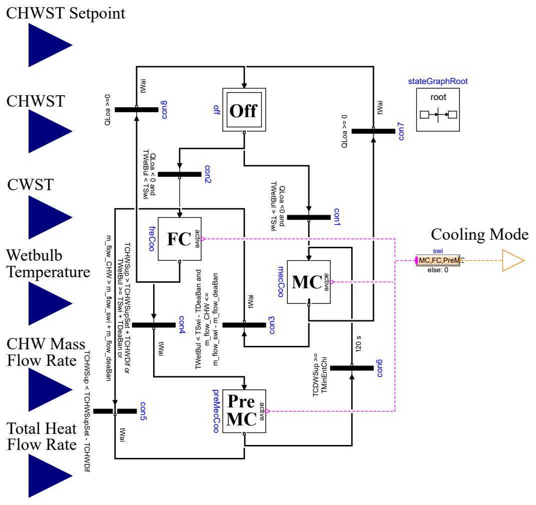


Figure 5. Diagram of the Modelica model for the master control.

ply temperature will be controlled. Further, the condenser water supply temperature setpoint changes between MC and Pre-MC modes. In FC mode, the chilled water supply temperature is controlled. If the measured temperature reading is greater than the setpoint plus the dead band, then the cooling tower fan PI controller is engaged to maintain the setpoint and the CW bypass valve is closed. If the measured temperature reading is less than the setpoint minus the dead band, then the cooling tower fans are off, the CW bypass valve opens, and the cooling tower isolation valves are controlled with the PI controller to maintain the setpoint. This control model is also instantiated on the top system model of the central plant.

Following the control logic of the real system, the three CHW pumps stage on/off based on the campus chilled water flow rate and the pump speeds (*Systems* level). The CHW pump speeds are modulated to maintain the pressure drop setpoint, with the pressure drop measured at the furthest connected building (*Units* level). For the two CW pumps, their staging is determined based on the cooling

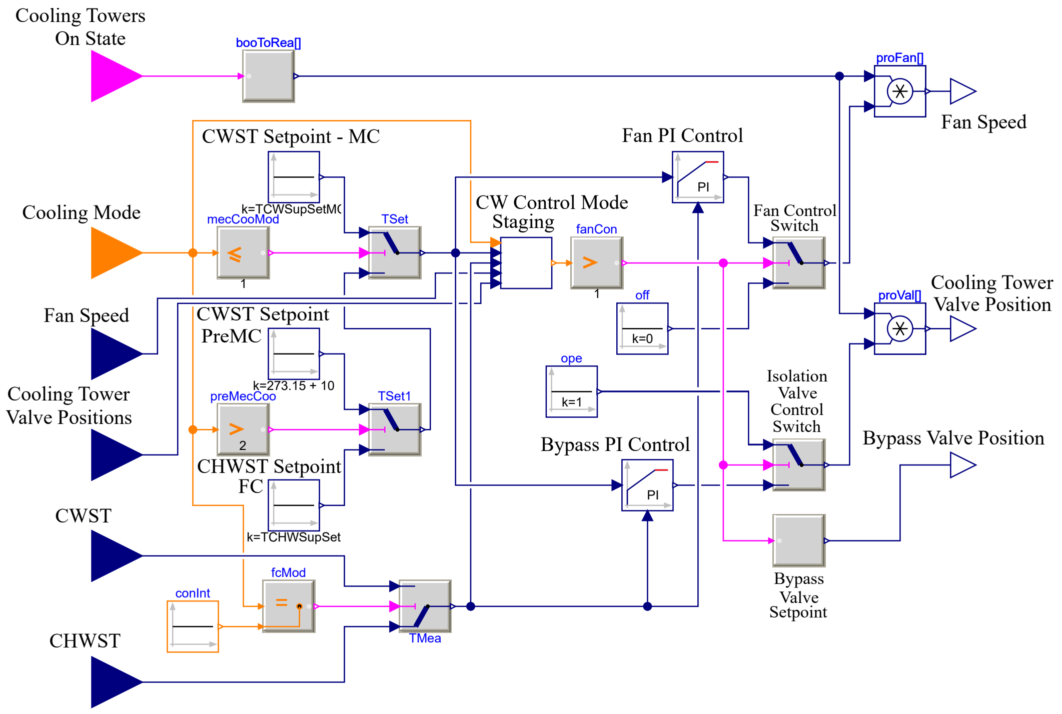


Figure 6. Diagram of the condenser water loop control.

mode and number of chillers running. Their flow rates are controlled at constant setpoints depending on the equipment running (1 chiller, 2 chillers, or 1 WSE).

Most *Devices* control was implemented with SISO PI controllers. For control and numerical stability, the proportional gain k and integral time constant Ti were tuned carefully. Consistent with past experiences in dynamic hydraulic models, we found values of $k = 0.1$ and $Ti = 120s$ were effective for most control valve applications. Stable pump and fan control parameters varied across the model.

3.3 Simulation Settings

All simulations ran in Dymola 2021 on Linux. While there are many suitable numerical solvers in Dymola for this type of application, CVODE (Hindmarsh, Serban, and Reynolds 2020) was selected for its suitability for solving stiff numerical problems (e.g., the system of differential algebraic equations contain both fast and slow dynamics, which make the selection of a variable time step size difficult for the solver), and in our experience, it typically simulates thermo-fluid systems quickly and robustly. All simulations ran using a tolerance of 1e-6. The computer contained 32 GB of RAM.

4 Verification and Validation

To establish an accurate baseline model, we validated major cooling equipment and system-level operation with respect to the measured data. We evaluated the Coefficient of Variation of the Root Mean Square Error (CVRMSE) using hourly time steps as follows:

$$CVRMSE = \frac{\sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{N-1}}}{\bar{y}} \qquad (1)$$

where y_i is the individual measured data, $\hat{y}_i$ is the corresponding simulation-predicted data, $\bar{y}$ is the mean of the measured dataset, and N is the total number of datapoints.

Due to uncertainties and gaps in measured data, validation of the entire DC plant for one year of measured data was not possible. This is consistent with many past DC and chiller plant modeling endeavors (Oppelt et al. 2016; Fu et al. 2019). Thus, two time periods representing typical summer and winter conditions were selected to validate the model, encompassing both full and part load conditions. With hourly data, the CVRMSE needs to be within 30% for the model to be considered validated (ASHRAE 2014).

Chilled water heat flow, mass flow, supply temperature, and return temperature were used to validate the model based on the limited availability of historical measurements. Ideally, the pump, chiller, and fan power would be used to validate the model; however, these electrical data points were not available. Thus, we verified equipment and system-level performance with design documents and by consulting plant operators. Historical data was used to the full extent possible to validate the model.

The validation results are summarized in Table 1. The simulations fell within the 30% CVRMSE threshold for all locations. During the summer period, the plant operated in mechanical cooling mode with the chiller meeting the cooling demand. While the plant operated in free cooling

Table 1. Validation results targeting CVRMSE less than 30%.

Equipment/	*CVRMSE (%)*			
System	$\dot{Q}_{CHW}$	$\dot{m}_{CHW}$	T_{CHWS}	T_{CHWR}
Summer Period (Aug. 1-14, 2018)				
Plant	18.8	12.9	8.9	10.3
Chiller	28.2	18.5	10.3	8.3
Winter Period (Jan. 28- Feb. 11, 2018)				
Plant	14.6	3.1	6.2	11.3
WSE	15.6	3.1	11.3	7.1

mode with the WSE meeting the entire cooling demand during the winter period.

Figure 7 visualizes the primary chiller's validation results. Upon inspection, the simulated CHW mass flow rate and supply and return temperatures match the measured data well. However during the nighttime, the measured CHW outlet temperature drifts below the minimum allowable value per the control specifications. Contrarily, the CHW outlet temperature is well controlled at the desired setpoint in the simulation. It is unknown why the real system does not maintain the CHW outlet temperature, and while it is undesirable from a control standpoint, it may be unavoidable due to the real system's transients and extraneous system requirements not included in the model. Based on these validation results, the accuracy of the DC plant model is within acceptable limits of the real system's measured data and expected performance.

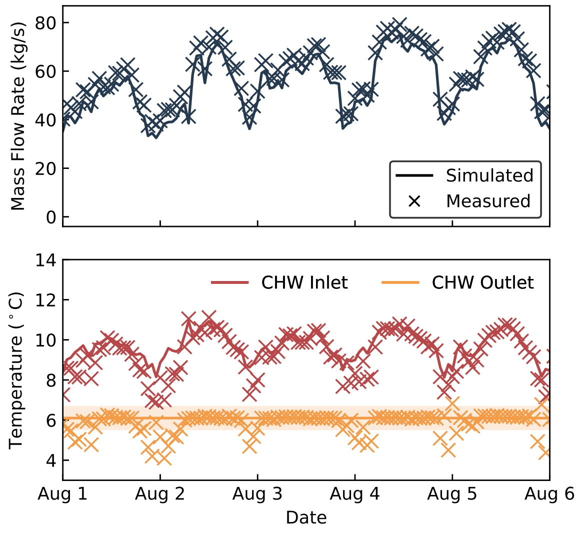


Figure 7. Primary chiller validation results in early August with the highlighted region indicating the control limits for the CHW leaving temperature.

5 Optimization Problem

We formulated a sequence of single objective optimization problems that, collectively, minimize the plant's annual energy consumption. The sequence of problems were formulated as follows. Let $\tau = 1$ year and $M \in \mathscr{N}$ be the number of intervals over which the optimization problem was solved. Then, we solved the set of problems $\mathbf{P}_i$, with

$$\mathbf{P}_i \quad \min_{x \in [\underline{T}_{CWS}, \overline{T}_{CWS}]} E_{Pla,i}(x), \qquad (2)$$

$$E_{Pla,i}(x) = \int_{t_i}^{t_{i+1}} (P_{CH}(x,s) + P_{CWP}(x,s) + P_{CHWP}(x,s) + P_{CT}(x,s))ds$$

with $t_i \in \{t_i \in \mathscr{R} \,|\, t_i = i\tau/M, i \in \{0,\dots,M-1\}\}$, where the independent variable x is CWST setpoint, $E_{Pla,i}$ is the total plant energy during the optimization period $t \in [t_i, t_{i+1})$, P_{CH} is the power of the chillers, P_{CWP} is the power of the condenser water pumps, P_{CHWP} is the power of the chilled water pumps, P_{CT} is the power of the cooling towers, $\underline{T}_{CWS}$ is the condenser water supply temperature low limit, and $\overline{T}_{CWS}$ is the condenser water supply temperature high limit. Through this method, the CWST setpoint is selected for each interval (e.g., there are 365 setpoints for a daily optimization case with $M = 365$).

Based on the chiller's specification documents, the condenser water supply temperature low and high limits are $10.0°C$ and $29.4°C$, respectively. These are used through the optimization process.

Optimization problems with time horizons of one day, week, month, and year are solved using the Optimization library version 2.2.4 (Pfeiffer 2012). Released alongside Dymola 2021, this library allows multi-objective optimization of complex systems within Dymola's modeling and simulation environment. The user interface allows for quick formulation of optimization problems, while the model's state values can be reinitialized for consecutive optimization runs without needing to rerun the entire optimization. For numerical optimization algorithms, we employed the simplex method due to it quicker computational speed as a local method and suitability for handling functions that are not smooth. Optimization and simulation tolerances of 1e-5 and 1e-6 respectively are used for all cases.

6 Results

For all cases, the optimized CWST setpoint and energy savings followed similar trends (Figure 8). Due to the limited number of MC hours in winter and fall seasons, the optimized CWST setpoint often stayed at the current setpoint during these times. The optimized CWST setpoint during MC mode was generally above the current setpoint for all cases.

The CWST optimization reduced the plant's annual energy consumption under all time horizon cases (Table 2), while still meeting the cooling loads at the building end

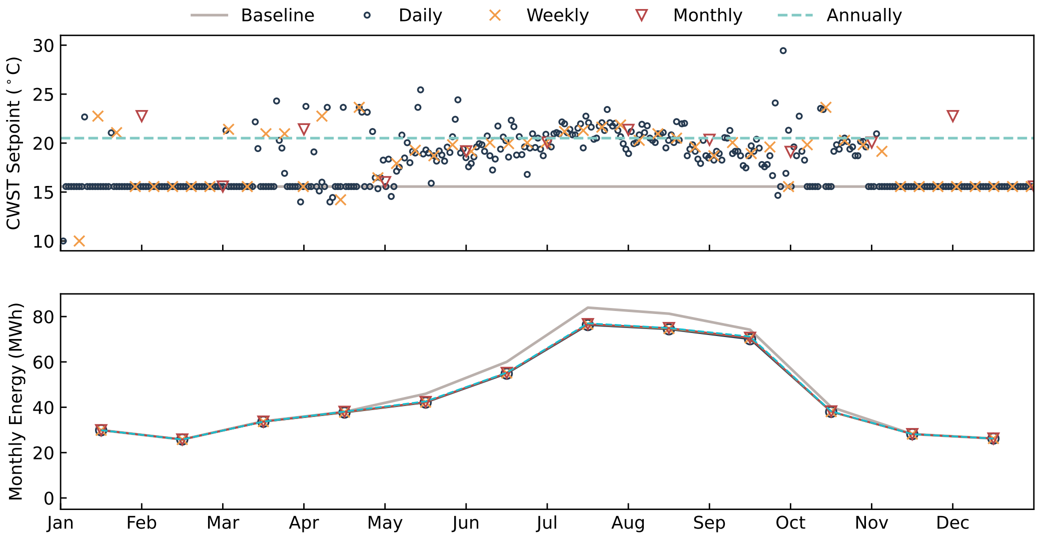


Figure 8. Condenser water supply temperature optimization results across multiple time horizons.

Table 2. Condenser water supply temperature optimization results across multiple time horizons. Energy values represent the plant's annual site energy, and savings are relative to the plant's current implementation (baseline).

Optimization	*CWST (°C)*		*Energy*	*Savings*
Time Horizon	*Mean*	*SD*	*(MWh)*	*(%)*
No optimization (baseline)	15.6	N/A	567.5	–
Daily	17.9	2.7	536.8	5.4
Weekly	18.6	2.9	538.9	5.0
Monthly	19.5	2.6	539.6	4.9
Annually	20.5	N/A	541.0	4.7

users. Annual energy savings were achieved from 4.7% (annual optimization) to 5.4% (daily optimization). Because the CWST is controlled in MC mode while the CHWST is controlled in FC mode, the energy savings from CWST optimization occurred during MC mode only. During mechanical cooling, the annual energy savings ranged from 7.4% with annual optimization to 8.6% with daily optimization.

7 Conclusion

Modeling and simulation of DC systems present ample opportunities for energy-efficient cooling systems at district scales. While Modelica is promising for this application, research in this area is still generally lacking, particularly for central plants featuring free cooling from WSEs. This work aimed to fill this gap by demonstrating how the new models contributed to the open-source Modelica Buildings library can be used for detailed energy analysis and optimization of a DC plant with a WSE connected in parallel with the chillers.

Through CWST optimization cases, around 5% plant energy was saved with minimal improvements achieved by decreasing the optimization time horizon. This indicates that the seasonal variation on daily through monthly scales does not greatly affect the optimization results, reconfirming the results achieved in previous studies (Huang, Zuo, and Sohn 2017). We recommend that the plant implement the annual CWST optimization because it is a robust and simple control retrofit.

The CWST optimizations exemplify retrofit strategies that are possible with the detailed Modelica models, but are by no means comprehensive. In the future, we plan to pursue additional retrofit strategies with higher energy saving potentials, including integrating the WSE with the chillers, adding thermal storage, and integrating the high-fidelity plant model with a complete district model to evaluate co-operational strategies across buildings and the plant.

Acknowledgements

This work was supported by the Building Technologies Office of the U.S. Department of Energy, under contract numbers DE-AC36-08GO28308 and DE-AC02-05CH11231. This research was also supported by the National Science Foundation under Award No. IIS-1802017. BIGDATA: Collaborative Research: IA: Big Data Analytics for Optimized Planning of Smart, Sustainable, and Connected Communities. Further, 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. The authors would also like to thank the campus facilities team for their assistance with data collection, expert advice, and overall support of this project.

References

Anderson, Austin, Behnaz Rezaie, and Marc A. Rosen (2021). "An innovative approach to enhance sustainability of a district cooling system by adjusting cold thermal storage and chiller operation". In: *Energy* 214, p. 118949. ISSN: 03605442. DOI: 10.1016/j.energy.2020.118949.

ASHRAE (2013). "District Cooling Guide". In: *District Cooling Guide*. Atlanta, GA: American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc. ISBN: 9781936504428.

ASHRAE (2014). *ASHRAE Guideline 14-2014: Measurement of Energy, Demand, and Water Savings*. Tech. rep. Atlanta, GA.

Chow, Tin Tai et al. (2004). "Energy modelling of district cooling system for new urban development". In: *Energy and Buildings* 36.11, pp. 1153–1162. ISSN: 03787788. DOI: 10.1016/j.enbuild.2004.04.002.

del Hoyo Arce, Itzal et al. (2018). "Models for fast modelling of district heating and cooling networks". In: *Renewable and Sustainable Energy Reviews* 82.July, pp. 1863–1873. ISSN: 18790690. DOI: 10.1016/j.rser.2017.06.109.

Falay, Basak et al. (2020). "Enabling large-scale dynamic simulations and reducing model complexity of district heating and cooling systems by aggregation". In: *Energy* 209, p. 118410. ISSN: 03605442. DOI: 10.1016/j.energy.2020.118410.

Fu, Yangyang et al. (2019). "Equation-based object-oriented modeling and simulation for data center cooling: A case study". In: *Energy and Buildings* 186, pp. 108–125. ISSN: 0378-7788. DOI: 10.1016/j.enbuild.2019.01.018.

Gang, Wenjie et al. (2015). "Performance Assessment of District Cooling System Coupled with Different Energy Technologies in Subtropical Area". In: *Energy Procedia* 75, pp. 1235–1241. ISSN: 18766102. DOI: 10.1016/j.egypro.2015.07.166.

Hindmarsh, Alan C, Radu Serban, and Daniel R. Reynolds (2020). *User documentation for CVODE v5.1.0 (SUNDIALS v5.1.0)*. Tech. rep. Lawrence Livermore National Laboratory. URL: https://computing.llnl.gov/projects/sundials/sundials-software.

Huang, Sen, Wangda Zuo, and Michael D. Sohn (2017). "Improved cooling tower control of legacy chiller plants by optimizing the condenser water set point". In: *Building and Environment* 111, pp. 33–46. ISSN: 03601323. DOI: 10.1016/j.buildenv.2016.10.011.

Hydeman, Mark and Guo Zhou (2007). "Optimizing chilled water plant control". In: *ASHRAE Journal* 49.6, pp. 44–54. ISSN: 00012491.

ICF LLC and International District Energy Association (2018). *U.S. District Energy Services Market Characterization*. Tech. rep. Washington DC: U.S. Energy Information Administration. URL: https://www.eia.gov/analysis/studies/buildings/districtservices/pdf/districtservices.pdf.

International Energy Agency (2018). *The Future of Cooling: Opportunities for energy-efficient air conditioning*. Tech. rep., p. 92.

International Energy Agency (2019). *2019 Global Status Report for Buildings and Construction: Towards a zero-emissions, efficient and resilient buildings and construction sector.* Tech. rep., pp. 1–48.

Karami, Majid and Liping Wang (2018). "Particle Swarm optimization for control operation of an all-variable speed water-cooled chiller plant". In: *Applied Thermal Engineering* 130, pp. 962–978. ISSN: 13594311. DOI: 10.1016/j.applthermaleng.2017.11.037.

Ling, Li et al. (2018). "Energy saving analysis of the cooling plant using lake water source base on the optimized control strategy with set points change". In: *Applied Thermal Engineering* 130, pp. 1440–1449. ISSN: 13594311. DOI: 10.1016/j.applthermaleng.2017.10.152.

Marafeq Qatar (2015). *District Cooling: GCC and Qatar*. Tech. rep. April.

Matsouka, Kenichi and David Hill (2020). "Online Optimization of Cooling Water System in a District Cooling Plant by Using Digital Twin". In: *ASHRAE Transactions* 126.2, pp. 427–434.

Oppelt, Thomas et al. (2016). "Dynamic thermo-hydraulic model of district cooling networks". In: *Applied Thermal Engineering* 102, pp. 336–345. ISSN: 13594311. DOI: 10.1016/j.applthermaleng.2016.03.168.

Pfeiffer, Andreas (2012). "Optimization Library for Interactive Multi-Criteria Optimization Tasks". In: *Proceedings of the 9th International Modelica Conference*. Vol. 76. Munich, Germany, pp. 669–680. DOI: 10.3384/ecp12076669.

United States Department of Energy (2018). *EnergyPlus Version 8.9.0 Documentation: Engineering Reference*. Tech. rep., p. 1716. URL: https://energyplus.net/sites/all/modules/custom/nrel_custom/pdfs/pdfs_v8.9.0/EngineeringReference.pdf.

van der Heijde, Bram et al. (2017). "Dynamic equation-based thermo-hydraulic pipe model for district heating and cooling systems". In: *Energy Conversion and Management* 151, pp. 158–169. DOI: 10.1016/j.enconman.2017.08.072.

Wang, Lan, Eric W.M. Lee, and Richard K.K. Yuen (2018). "A practical approach to chiller plants' optimisation". In: *Energy and Buildings* 169, pp. 332–343. ISSN: 03787788. DOI: 10.1016/j.enbuild.2018.03.076.

Wetter, Michael et al. (2014). "Modelica Buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506.

Zabala, Laura et al. (2020). "Virtual testbed for model predictive control development in district cooling systems". In: *Renewable and Sustainable Energy Reviews* 129. ISSN: 18790690. DOI: 10.1016/j.rser.2020.109920.

Appendix

See Table 3 for the plant's nominal information.

Table 3. Nominal information for the central cooling plant equipment.

Equipment	*Qty.*	*Nominal Equipment Information*		*Unit*	*Value*
Chiller	2	Nominal Capacity		kW	2 450
		Design Efficiencies	Coefficient of Performance (COP)	–	6.80
			kW/ton	kW/ton	0.517
		Evaporator	Flow Rate	m^3/s	0.0878
			Pressure Loss	kPa	29.0
			Design Entering Temperature	°*C*	12.2
			Design Leaving Temperature	°*C*	28.4
		Condenser	Flow Rate	m^3/s	0.133
			Pressure Loss	kPa	64.6
			Design Entering Temperature	°*C*	23.3
			Design Leaving Temperature	°*C*	28.4
		Compressor	Number	–	1
			Speed Type	–	Variable
			Power	kW	366
Waterside Economizer	1	Nominal Capacity		kW	2 820
		Design Approach Temperature		°C	1.7
		Chilled Water Side	Flow Rate	m^3/s	0.121
			Pressure Loss	kPa	48.4
		Condenser Water Side	Flow Rate	m^3/s	0.151
			Pressure Loss	kPa	83.1
Chilled Water Pump	3	Head		kPa	252
		Power		kW	29.8
		Flow Rate		m^3/s	0.0883
		Speed Type		–	Variable
Condenser Water Pump	2	Head		kPa	338
		Power		kW	55.9
		Flow Rate		m^3/s	0.126
		Speed Type		–	Variable
Cooling Tower	2	Nominal Capacity		kW	2 813
		Nominal Flow Rate		m^3/s	0.158
		Number of Cells		–	2
		Nominal Fan Power		kW	22.4
		Fan Speed Type		–	Variable
		Design Temperatures	Hot Water	°C	28.2
			Cold Water	°C	22.6
			Wetbulb	°C	17.8

Long Term Technical and Economic Evaluation of Hydrogen Storage Technologies for Energy Autarkic Residential Complexes

L. Schindhelm[1] A. Vojacek[2] J. Brunnemann[2]

[1]Fraunhofer Institute for Chemical Technology, Pfinztal, Germany lucas.schindhelm@ict.fraunhofer.de
[2]XRG Simulation GmbH, Hamburg, Germany, {vojacek,brunnemann}@xrg-simulation.de

Abstract

We present an assessment of different types of hydrogen storages used as long term energy buffers for a local community complex of households in terms of economics and energy autarky. The models used in this study are partly based on the TransiEnt Modelica Library, which is being developed for the dynamic simulation of coupled energy supply systems with high shares of renewable energies. It turns out that dynamic simulations are mandatory in order to optimise the system parameters. Starting from a best case evaluation of a one year linear optimisation we develop a forecast based control logic of the whole energy system, including its physicalities. Based on our results, a storage consisting of pressurized gas bottles has proven to be the most favourite solution in terms of price and level of autarky. A liquid organic hydrogen carrier might be a competitive alternative for larger urban districts.

Keywords: Energy system, autarky, economic, long term, Hydrogen storage, Control logic, TransiEnt library, ClaRa library, Linear optimisation

1 Introduction

1.1 Context of this Project

The German Renewable Energy Law (BMU 2021) aims for 80 % of electrical and 60 % of primary energy supply from renewable energy sources (RES). The energy density with respect to surface area is magnitudes smaller for RES than for centralized energy systems like nuclear power plants. Energy transmission lines would have to be scaled to the massive volatile peak power output of RES, especially photovoltaics (PV), in order to transport energy e.g. from one part of the country to the other. Decentralized energy systems can be an alternative to reduce the costs and organizational difficulties of building additional transmission capacity by various means of local energy storage and smart energy management.

Funded by the 6th energy research program (BMU 2011) the joined research project "Energy Buffer" (EP) was carried out. In the course of this project a "hydrogen battery"(HB) comprising an PEM-electrolyser (ELY), a PEM-fuel cell (FC) (Proton Motor Fuel Cell GmbH 2021), an energy management (EMS) and a pressurized hydrogen storage (pressure level: 30 bar) was designed and a demonstrator build from market-available components. A modular design was applied to a passive house residential area in Stadtroda (Thuringia, Germany) (9 houses and one central facility) as the key reference point. The residential area is planned to be 99 % time- and energy quantity-autarkic (level of autarky $LoA_{time} = LoA_{energy} = 0.99$). PV is the exclusive energy source. The heat supply is based on a detailed prescribed concept (Frey 2019) and the passive house standard (PHI 2021).

The process of developing the project EP started as early as 2011. Recent commercial designs with similar parameters can be found in (Stiftung Umwelt Arena Schweiz 2021) and (HPS Home Power Solutions GmbH 2021).

Technological and economic evaluations of hydrogen storages in energy supply systems have been carried out earlier, e.g. in (Macagno 2004). In Modelica, the modeling and simulation of hybrid renewable energy systems containing among others PV and storages have been investigated in (Fritzson 2013), numerical implications of such models have been analysed in (Kofman 2016). In a recent work (Bentvelsen 2019) investigated a controllable electrolyser using OpenModelica. There a wind turbine provides electric power which is scheduled by a forecast based control algorithm between an industrial local grid and an electrolyser. A Modelica model of a fuel cell and a metal hydride storage has been developed in (Scarisbrick 2019), focusing on physical aspects of the components. A hydrogen production system for residential buildings has been modelled and investigated in (Henriquez 2018). There again the focus was put on the component modelling, while the control algorithm is not presented in detail.

While in the cited literature individual system components models have been created for the particular simulation study the work presented in this paper is based on standard components of the TransiEnt library (TransiEnt v1.2.0 2020), which are extended by certain physicalities, such as load depending efficiencies, minimum loads or limits on yearly start/stop cycles. These properties are relevant for operating the system using an underlying control logic, which is based on a 24h ahead weather and demand forecast model.

1.2 Outline of Paper

In subsection 2.1 the main results of a preliminary linear optimisation of the HB system are presented, which is based on certain idealisations of the real system. These results serve as an upper bound on the performance of the real system. The necessity of developing a dynamic model of HB in Modelica for a more extractable and realistic approach is developed in subsection 2.2. There we also elaborate on different available storage technologies and discuss pros and cons of two storage types that were finally analysed.

Used libraries and physicality-based models of system components that were newly developed in this work are shown in subsection 2.3. The steps of developing the crucial dynamic system logic and the effects of improving the logic regarding system performance in comparison to results from linear optimisation are presented in subsection 2.4. The control logic is then compared to the initial linear optimization (LO) results for a one year time period. We show that a 24h ahead forecast model is sufficient in order to bring the real system close to the idealised LO result, if physicalities are neglected. In section 3 the analysed setups of HB are evaluated in terms of their autarky level and costs. An assessment of the feasibility of HB in the temporary economic state and policies can be found in section 4. Usability and ability of the developed Modelica libraries and models are depicted in section 4 as well to conclude this work.

2 Preliminary Work

2.1 Linear Optimisation

Firstly, a linear optimisation approach for modelling was taken to further guide the development of a more detailed physical model in Modelica, in particular, it provided results of an ideal case. The components of the HB and PV have been modelled for the optimisation in oemof (Hilpert et al. 2018). Oemof uses mixed-integer linear programming (MILP) to mathematically optimise the system for minimising the electrical energy import from the superior grid. Heat supply and its electricity demand are modelled and simulated in TRNSYS (University of Wisconsin–Madison Solar Energy Laboratory 1975), where transient physical models are utilised under use of a case-specific control logic. The electricity demand of the heat supply is used as input data for the oemof simulation, whereas the waste heat of ELY and FC are input data for the TRNSYS simulation. In this way there is an implicit serial coupling between both tools.

2.1.1 Parameters of Physical Models and Input Data

In Table 1 the main parameter sets, that are used in the optimisation, are shown. Further, there were considered additional technical constraints for ELY and FC such as maximum start-stop cycles per annum 500/1500 and maximum runtime 2000/4500 respectively. The electrical demand, excluding heat supply, is extracted from real data of sonnen GmbH, processing nine individual houses' annual electrical consumption between 2-3 MWh/a. Note that ELY and FC units each consist of two independent modules in order to better cope with partial load. The optimisation time step is one hour.

Table 1. Characteristics of the system components.

Device	*Efficiency*	*Capacity*	*el. Power*
	%	*kWh*	*kW*
FC	47	-	4.7
ELY	51	-	5.8
hydrogen storage	-	13422 (400 kg)	-
lithium-ion batteries	93.3/91.6 (out/in)	106	33.3
PV	n.a.	-	90
heat pump	-	-	9 x 2.1

2.1.2 Interpretation of Optimisation Results

This section focuses on the optimisation results regarding capacity of the hydrogen storage, since the cost of the storage capacity is comparably high ($\sim$ 1000€ per kg H_2). Further, capacities and peak power values were determined within technical and regulatory limits. Reasonable engineering guesses and preliminary optimization were used (e.g. for number of FC modules etc.). The set of applicable devices was further limited to the product portfolio of the participating companies and by some budget restrictions. Hence, initial, quite rigid device choices had to be made, that are taken as granted in this study.

The optimisation results are in good agreement with the calculation of ideal Modelica model excluding physicalities (see Figure 6 in subsubsection 2.4.3) whose main outcome are presented in Table 3 and in Table 4.

Table 2. Level of autarky from LO in terms of energy and time.

Capacity of storage [kg]	LoA_{energy}	LoA_{time}
400	0.96	0.98
300	0.92	0.94
200	0.88	0.92

It is clearly visible that the storage capacity of 400 kg hydrogen shows the best performance regarding autarky (see Table 2). The PV supply gap and the high energy demand of the heat pumps in the winter months lead to a rapid decline of the state-of-charge (SOC). As a consequence, high amounts of hydrogen have to be stored and it is crucial to maximize the stored hydrogen mass in summer. For this, decisions, like storing an energy surplus in the batteries for short-term or in the HB for the long-term, have to be made. It is of great importance to maximize the productivity of FC and ELY without overstretching the operational constraints. This overstretching could easily happen, if e.g. ELY is always switched on if minimal energy surplus sufficient for its operation is detected.

ELY than could lose crucial hours of high productivity. In this optimisation study the solver has ex-ante knowledge of all input data sets, so the decision, e.g. when to run which module of ELY, is not reproducible in real conditions and extremely dependent on the highly specific input data. Since there is only a cost penalty for using the superior grid to cover the energy demand, each solver run can use the full range of a constraint limits and options, what may lead to inscrutable behaviour of the components. It is possible that one module is switched off for one hour and is then reactivated for the next hour, while the second module runs for the switched off hour. Another example of this artefact is the decision, when to start filling the tank. Especially, in the case of 200 kg hydrogen, it was observed that the systems only starts producing hydrogen later in the year. It is enough to fill the tank in this specific case, but dependencies of weather and real demand are not accounted for. Therefore, this behaviour is not desirable for the system while using a real process control and, consequently, not foreseeing the time profiles of demand, PV etc. on a one year time scope. To achieve this, a control system has to be implemented, that can manage short-term and long-term storage decisions.

In order to test and validate the optimisation results under real conditions, XRG Simulation GmbH received the task to develop such a process control logic and to simulate the system in Modelica using the same input data as processed in linear optimisation.

2.2 Request for a Modelica System Model

By using a dynamic Modelica System Model it is possible to implement the requested more dynamic and comprehensible (logic) model of the demonstrator. Following tasks and aspects are adressed in the Modelica System Model to add significant value to the results of the linear optimisation:

- implementing an extractable distribution logic of electric power depending on time and load (ELY-FC-grid-batteries)
- inclusion of on/off cycles of components and their physical constraints, i.e. start-up / shut down delays, minimum/maximum loads and battery capacity
- avoidance of faulty switching (artefact from linear optimization) on/off modules of ELY and FC
- analyzing of different H_2-storage technologies, especially utilising improved physicalities
- optimisation of the system parameters and its control logic under real operation

Consequently, a Modelica model, was developed, which shall describe the energy system at a higher degree of physical precision, while still performing with efficient calculation time simulating one calendrical year.

2.3 Component Model Library

TransiEnt v1.2.0 (2020) Modelica library was chosen as starting point for modelling. It already includes a comprehensive collection of models to describe and analyse integrated energy systems with high share of renewable energies according to environmental and economic aspects. The library was developed in the research project TransiEnt.EE and its successor project ResiliEnt.EE (2021). The TransiEnt library uses the ClaRa v1.3.0 (2020) Modelica library which allows dynamic simulation of thermal hydraulic energy systems such as power plants, thermal storages etc. This library combination creates a comprehensive and powerful tool for modelling local energy systems and power plants. All supplement models developed within this project have been put together in a Modelica library. It consists of models of control logic, ELY, FC, storages, compressor, separator, cost models of key components and additionaly handy models such as sensors for measurement of power on ElectricPowerPort and calculation the cost to/from grid etc.

Selection of applicable H_2-storage technologies. In order to select applicable storage technologies to be modelled an extensive literature study (HydrogenEurope 2020; Hydrogenious 2020; FuelCellStore 2020) was performed. It evaluates the current state of the art of hydrogen storage technology for small scale residential areas. The study revealed two preferable storing techniques for hydrogen: *pressurized* and liquid organic carrier *LOHC*. Pros ("+") and cons ("-") of these technologies are:

Pressurized Storage System

+ most common hydrogen storage technology
+ very simple release mechanism, minimal complexity at customer site
+ price relatively low
- heavy system due to pressurised components (steel pressure tanks/bottles)
- low storage density
- different pressure levels used for different applications

Within this paper we consider 28 bars tanks (approx. 2 kg/m^3 at 20°C) and 200 bars bottles (approx. 14 kg/m^3 at 20°C).

Liquid Organic Hydrogen Carrier (LOHC)

+ demonstrator units already built (Hydrogenious 2020) with parameters sufficient for this project.
+ H_2 stored at ambient condition (temperature, pressure), hardly inflammable, non-explosive
+ light weight system (i.e. storage in plastic canisters)
- lower overall efficiency (energy required for releasing H_2 (approx. at 300°C) from LOHC, at times when there is no excess of PV power)

Other approaches of storing the hydrogen are currently either too expensive, too complex for a small scale urban place, high energy demanding, technically immature or inquiring specific condition of the underground such as e.g. salt caverns, exhausted oil and gas fields etc. These storage systems are: *Liquefied H_2; Cold- and cryo-compressed H_2; Material-based H_2 storage; Hydride storage systems; Surface storage systems and Underground storage*. Consequently, these systems have not been further evaluated in this work.

Model of ELY&FC&Pressurized Storage. The model of ELY, FC and H_2 pressurized storage was implemented as one lumped component in very simple manner. The mass $m_{H_2}$of H_2 in the storage was balanced according to Equation 1.

$$\frac{d}{dt} m_{H_2} = -\frac{1}{\mathtt{LHV}_{\mathtt{H_2}}} \left(\eta_{ELY} P_{ELY}^{(el,set)} + \frac{P_{FC}^{(el,set)}}{\eta_{FC}} \right) \quad (1)$$

Here $\eta_{ELY} = 0.51$ and $\eta_{FC} = 0.47$ are the efficiencies of ELY and FC. Moreover $P_{ELY}^{(el,set)}$ and $P_{FC}^{(el,set)}$ are their electric power input values. The model assumes the lower heating value $\mathtt{LHV}_{\mathtt{H_2}}$ of H_2 as 120 MJ/kg, i.e. $\mathtt{LHV}_{\mathtt{H_2}}$=33.33 kWh/kg. The model was further elaborated to capture time dependent efficiencies, modularity (2 ELYs and 2 FCs), stand by losses of ELY/FC, simple compressor, pressure dependent storage etc.

Model of Liquid Organic Hydrogen Carrier Storage. LOHC is a good alternative to pressurized H_2 storages as pointed out before. Demonstration units have been built (e.g. by Hydrogenious (2020) with storing capacity $\dot{m}_{H_2}$ = 9.1 kg/h and $\dot{m}_{H_2}$ = 3 kg/h releasing capacity. The maximum parameters of the exemplary households are 0.2 kg/h and 0.6 kg/h of $\dot{m}_{H_2}$ in ELY and FC respectively. Hence, the the size of the demonstration unit would be sufficient for our project.

The previous lumped model of ELY, FC and storage of H_2 (Equation 1), is adapted to capture dehydrogenating specific enthalpy $h^{(dh)}$, needed for extracting H_2 out of the LOHC.

$$\frac{d}{dt} m_{H_2} = -\left(\frac{\eta_{ELY} P_{ELY}^{(el,set)}}{\mathtt{LHV}_{\mathtt{H_2}}} + \frac{1}{\mathtt{LHV}_{\mathtt{H_2}} - h^{(dh)}} \frac{P_{FC}^{(el,set)}}{\eta_{FC}} \right) \quad (2)$$

According to (Krieger 2019) heat up of the liquid (around 300°C) for H_2 extraction (dehydrogenation) is realised by burning of H_2. A similar amount of heat is released (at slightly lower temperature around 250 °C) during the storage of H_2 (hydrogenating). Following Hydrogenious (2020), approximately 10 kWh of heat are needed in order to release 1 kg of H_2. Hence, $h^{(dh)}$ was set to $h^{(dh)}$ =10 kWh/kg. Taking into account $\mathtt{LHV}_{\mathtt{H_2}}$=33.33 kWh/kg, the overall efficiency of the process, electric power → H_2 → electric power, for LOHC can be evaluated to η_{LOHC}. It is only 0.165 (Equation 3) compared to 0.25 for pressurized technology (Equation 4).[1] The overall efficiency of LOHC (0.165) corresponds to findings in (Krieger 2019). This significantly reduces the attractiveness of LOHC storage, unless the heat produced during hydrogenation is utilized for the dehydrogenation.

$$\eta_{\text{LOHC}} = \eta_{\text{ELY}} \cdot \eta_{\text{FC}} \cdot \left(1 - \frac{h^{(dh)}}{\mathtt{LHV}_{\mathtt{H_2}}} \right) \quad (3)$$

$$\eta_{\text{Pressurized}} = \eta_{\text{ELY}} \cdot \eta_{\text{FC}} \cdot \eta_{\text{comp}} \quad (4)$$

This is illustrated in Figure 6. The LOHC H_2 storage runs empty more than two weeks earlier compared to the pressurised storage. Figure 7 presents a comparison from an el. energy flow perspective. It is evident that LOHC needs to take more electric power from the grid for a longer period of time.

Model of H_2 Drying Process. The aim of modelling the H_2 drying process was to estimate the energy consumption of that physical process. The model considers the drying of water-vapor saturated H_2, which is produced in ELY at 50-70 °C, to 200 ppm H_2O which fulfils the FC-requirement of 500 ppm H_2O with sufficient margin. The actual ELY available for this project uses a downstream zeolite filled adsorber. The adsorber material has to be exchanged or regenerated (dried out) every 200 hours of operation. This is certainly not acceptable for a continuous long term operation.

Therefore a more useable drying unit based on a condensation/freezing process which is common in practice (Bensmann et al. 2016; Tjarks et al. 2018; Kopp et al. 2017) was modelled in Modelica. The core of the model is a separator (XRG Simulation 2021). It is designed as a pipe that contains a water tank where liquid water is collected. The separator is surrounded by pipes that are filled with cooling liquid. The incoming H_2 gas is cooled down in order to decrease its saturation water content until liquid water occurs and freezes. The cooling liquid can also be used to reheat the frozen water.

The H_2 drying was added to the energy system model in order to evaluate the electric power consumption of this process. It was observed that electrical consumption of the exemplary households rises due to the drying process of H_2 just by 70 kWh/year (~0.2 kWh/kg H_2), which is ~0.5 % of the overall generated LHV_{H2} or ~0.2 % of total consumed el. power/year.

System Models. The overall model of the system has been developed at different levels of complexity. Firstly, a simple model with no logic has been created in Modelica, entirely from TransiEnt library components.

It uses all boundary conditions given from the linear optimisation results (such as electricity production from PV, electricity demand of households, electricity flow to/from batteries, electricity production from FC and electricity demand by ELY). Only the electricity flow from/to

[1] Efficiency of the considered compression work η_{comp} in the overall energy system is close to 1.

the grid is let free to be calculated. The boundary conditions are specified by hourly based time tables supplied by a text file. The model is connected to a lumped grid model, including primary and secondary control models. It includes a lumped generator model to mimic the synchronous grid of Continental Europe (UCTE).

In the final system model, (see Figure 1), fixed boundary conditions for usage of ELY/FC and batteries were replaced by physical models and corresponding control logic. The system model plus the control logic were kept the same for all storage technologies.

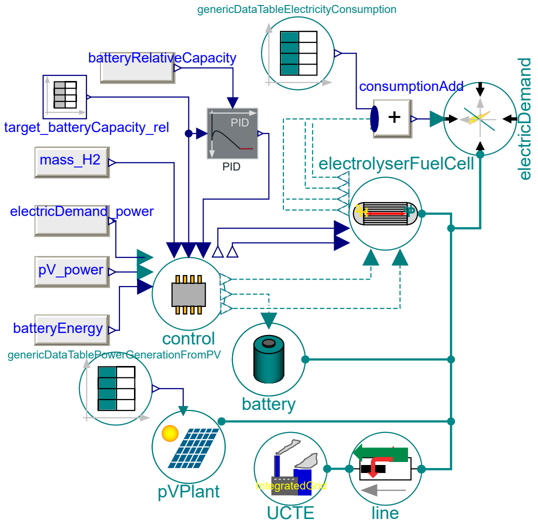


Figure 1. Final model of the energy system of households with control logic.

Resulting Objectives. Using the described inputs and models, the goal is to perform a technical and economic evaluation of pressurized and LOHC based H_2 storage technologies for three different H_2 storage capacities {200,300,400} kg for 30 years of operation. In particular we have to tackle the following tasks:

1. Re-examine the results of linear optimisation: are they consistent with the physical model?
2. Develop a control logic of the energy system storage in order to answer e.g.
   (a) Shall excess power produced from photovoltaic (PV) be firstly stored in batteries (if they have capacity available) or shall it be used to fill up H_2 storage?
   (b) If the two modules of ELY/FC shall operate, can/shall battery in some scenario supply power to ELY to charge H_2 storage?
3. Capture physical constraints (physicalities)
   (a) ELY/FC: startup time (heat up and lower efficiency), efficiency, power consumption during standby mode, two separate modules, each with minimum operation power
   (b) compressor for pressurised bottle storage
   (c) pressure dependent storage capacity
   (d) power consumption of hydrogen drying process (condensation/freezing)
4. Add weather and consumption forecast model into control logic.
5. Incorporate a cost model for all H_2 system components in order to predict the costs over 30 years.

2.4 Creating a Detailed Control Logic

In the sequel, we describe the emergence of the control logic as a step by step iteration benchmarked by the linear optimisation result. The initial simple Modelica model is further enhanced towards a more sophisticated logic and the previously described objectives.

2.4.1 Re-examination of Linear Optimisation Result

Figure 2 gives a comparison of the resulted energy flow to/from grid [2] by the simple system model of paragraph 2.3 and the linear optimisation model (see subsection 2.1) for a one year evaluation. There is a clear agreement of the results. Although, a slight deviation (max. 6%) can be observed, which is caused by hourly sampling of the linear optimsation: The values assumed constant by linear optimisation are not necessarily constant throughout 1 hour in the dynamic model.

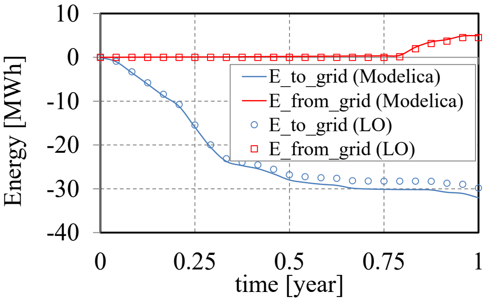


Figure 2. Comparison of el. energy flow results from a Modelica and linear optimisation model with no logic for 200 kg H_2 storage.

Simulation starts in the beginning of March which brings more sunny days and the PV power exceeds the household consumption together with battery and ELY capacity. This results in a continuous rise of *Energy put to the exterior electric grid* until October ($\frac{2}{3}$ of the year). The *Energy taken from the grid* starts to dominate from around November ($\frac{3}{4}$ of the year) as soon as the H_2 storage is emptied and the power of PV is almost zero.

2.4.2 Model with Simple Logic

A next step was the development of a very first simple logic for distribution of energy flows between grid, batter-

[2] *Energy from a grid* has a positive sign and *Energy to a grid* has a negative sign.

ies, ELY and FC. The logic [3] is depicted in Figure 3.

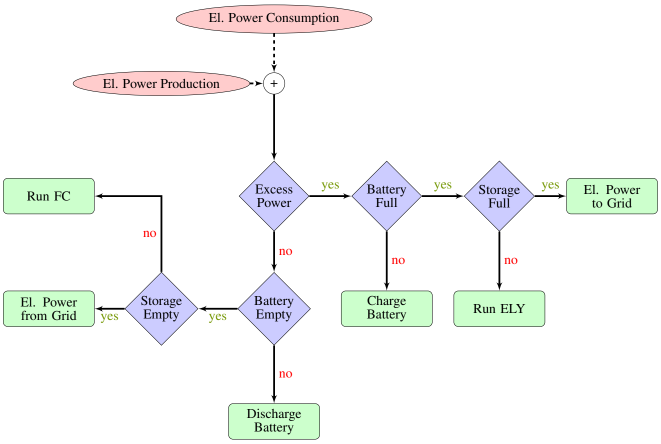


Figure 3. First simple control logic.

This logic is plugged into the simple overall energy system model (paragraph 2.3) and replaces fixed boundary conditions of battery, ELY and FC by realistic components, as shown in Figure 1. Hence, it uses electricity production from PV and electricity demand of households as table based bounday condition and the rest is calculated, i.e. electricity flow to/from batteries, electricity production from FC and electricity demand by ELY, electricity flow to/from grid). The simple logic model further assumes 1 lumped battery module, 1 lumped FC module and 1 lumped ELY module.

Figure 4 reveals the limitations of the simple logic. Maximum mass of H_2 in storage that is possible to reach with this simple logic is only 230 kg, while the results from linear optimisation show 400 kg. It reveals much lower H_2 production (of Modelica simulation) which results in a clear lack of H_2 in the storage already in the beginning of December ($\frac{5}{6}$ of the year). Clearly this is caused by the fairly immature control logic applied for initial implementation. Hence, further enhancement of the control logic was required.

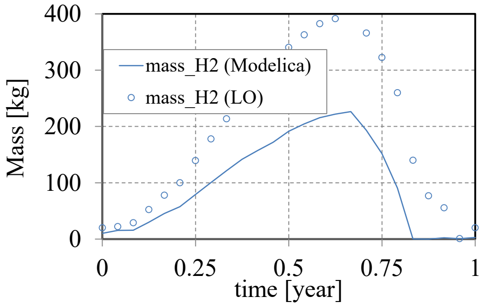


Figure 4. Comparison of resulted mass of H_2 in storage from a Modelica and linear optimisation model with simple logic.

It was found that the capacity of the battery stays most of the time in its upper range and frequently reaches its maximum and thus limits its further usage. This is different from the linear optimisation result and indicates an important direction for improvement.

2.4.3 Model with Improved Logic

Based on the previous unsatisfactory behaviour of the energy distribution in the system, an improved logic was worked out, see Figure 5. It was based on the simple logic and several extra features were added.

The first enhancement. The battery usage is optimised such that if battery is able to unload, i.e. has capacity, it shall distribute its power between household consumption demand and ELY. This is controlled according to a PID controller, see Figure 1. The PID controls the capacity level of the battery based on a *Forecast model* which computes target charge capacity for batteries. The principle of the *Forecast model* is as follows: The model looks 24 hours ahead for a predicted power consumption $P_{con}(t)$ of households and assumed power production from PV $P_{PV}(t)$. The integrated sum for upcoming 24 hours of the two above-named powers gives the required needed capacity for the batteries E_{store} as given in Equation 5. The relative $E_{storeRel}$ is defined as in Equation 6 where $E_{batteryNom}$ is the nominal battery capacity (e.g. maximum battery capacity). Then one can derive the relative target capacity of battery $E_{batteryTarget}$ according to Equation 7 using limited $E_{storeRel}$. The $E_{batteryTarget}$ is then kept constant for each upcoming 24 hours. In the simulations, the minimum value of $E_{batteryTarget}$ was set to 0.35 in order to have some margin for electricity demand needs. Otherwise, electricity from the grid would be frequently unintentionally used. Finally, the control logic contains a PID controller that keeps relative battery storage $E_{batteryRel}$ close to the value of $E_{batteryTarget}$ by using the free capacity of battery to charge ELY. The $E_{batteryRel}$ is prescribed as in Equation 8.

$$E_{store} = \int_{t_0}^{t_0+24h} (P_{con}(t) + P_{PV}(t))\, dt \tag{5}$$

$$E_{storeRel} = \frac{E_{store}}{E_{batteryNom}} \tag{6}$$

$$E_{batteryTarget} = 1 - \min(1, \max(0, E_{storeRel})) \tag{7}$$

$$E_{batteryeRel} = \frac{E_{battery}}{E_{batteryNom}} \tag{8}$$

The second enhancement. The logic is enabled to simultaneously run ELY and charge batteries if there is enough excess power.

The key set of rules for the improved control logic can be summarized as follows:

- *Charging of battery*:
  Battery is charged if there is excess of power while battery is not yet full. The charging power is derived as excess power minus the power taken by ELY.

[3] If max. power of the components is reached then the excess power is taken or send to the grid.

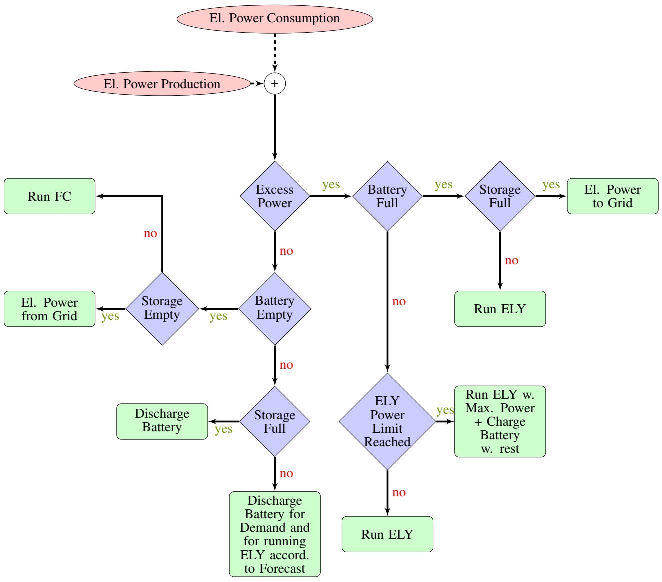


Figure 5. Advanced control logic.

- *Discharging of battery*:
  Battery is discharged for a load demand or for the ELY if there is a load demand and battery and H_2 storage is not full.
- *FC usage*:
  FC is used if there is a load demand and battery is empty and H_2 storage is not empty.
- *ELY usage*:
  ELY is used if there is excess power and battery is full and storage is not full. Further, use ELY if there is excess power to simultaneously charge ELY and battery and if there is enough power to charge both ELY and battery (otherwise only ELY, i.e. ELY has priority). Finally, use ELY from battery power, but only if there is net demand for power in the system (otherwise undesired charging of battery with simultaneous discharging of battery for ELY would appear). It would go against each other.

Resulting system operation. Finally, with this improved logic, a maximum of 400 kg of H_2 can be reached as presented in the Figure 6 (curve labelled as *ideal*) which is in agreement with the linear optimisation analysis. Subsequently, the battery usage is enhanced. However, what the Figure 6 additionally shows is that when further physical constraints (subsection 2.4), to mimic realistic behaviour of the system, is implemented, H_2 mass reduces dramatically. For example, the maximum of H_2 mass in storage barely reaches 340 kg, i.e. just 85 % of the idealized scenario. The LOHC process only reaches 300 kg, due to the worse efficiency of the process. In Table 3, the main results are shown for 400 kg pressurized H_2 storage for ideal (no physicalities) and real (with physicalities) case.

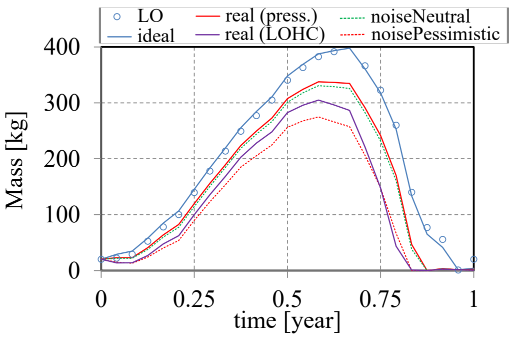


Figure 6. Comparison of resulted mass of H_2 in storage from Modelica and linear optimisation (LO) model with improved logic for ideal (no physicalities) and real (with physicalities described in subsection 2.4) pressurised+LOHC case together with real pressurised case with neutral and pessimistic noise.

Table 3. Overview of results for pressurized 400 kg H_2 storage for ideal (no physicalities) and real (with physicalities) case.

Quantity	$Value_{ideal}$	$Value_{real}$
El.power$_{fromGrid}$	1.6 MWh	3.8 MWh
El.power$_{toGrid}$	17.3 MWh	16.7 MWh
Start/Stop ELY	{256/256}/a	{278/277}/a
Start/Stop FC	{136/136}/a	{223/223}/a
Runtime ELY	{3043/3040}h/a	{2814/2813}h/a
Runtime FC	{1603/1585}h/a	{1332/1330}h/a

Note that ELY and FC units each consist of two independent modules (two numbers in brackets in Table 3) and for the case with physicalities, the minimum power limitation of one unit causes more start/stop cycles. However, it remains well below the limits.

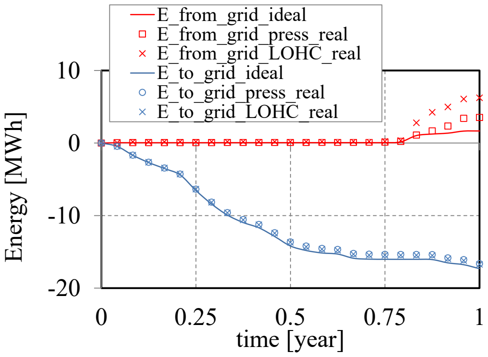


Figure 7. Comparison of el. energy flow results of a ideal pressurized 400 kg H_2 case (no physicalities) together with pressurized real and LOHC technology (with physicalities).

Figure 6 further demonstrates the functionality of the logic in case of *forecast errors*: Random noise is added to the original boundary condition of the real case. The *noisePesimistic* case introduces random noise to power consumption with range (0% to +20% of original) and noise to PV production (0% to -20% of original values). Similarly in the *noiseNeutral* case the range was chosen for power consumption and PV production (±20% of original values).

Variation of geographic location Our system model was further extended to consider different geographic locations. The reference site was Jena (Germany) and we chose one northern and one southern European city, i.e. Copenhagen (Denmark) and Marseille (France) respectively. A model from HumanComfort Modelica library HumanComfort v.2.11 (2020) was used in order to calculate the sun position dependent on the location and solar time. Together with the maximal possible solar irradiance (1367 W/m^2) the Extraterrestrial irradiance on horizontal earth´s surface (G$_{extHor}$) is calculated for each location for the whole one year period. The value of G$_{extHorRef}$ for Jena was used as a base and the scaling factors for the two other locations (r$_{Copenhagen}$ for Copenhagen) were derived as a ratio of the day integrals of the corresponding G$_{extHor}$ values as follows:

$$r_{Copenhagen}(t) = \frac{\int_t^{t+24h} G_{extHorCopenhagen}(\tau)d\tau}{\int_t^{t+24h} G_{extHorRef}(\tau)d\tau} \tag{9}$$

and r$_{Marseille}$ for Marseille was derived in similar manner. Using these ratios the reference electricity PV production for Jena was scaled for the other locations. Finally, $E_{batteryTarget}$ (Equation 7) was recalculated (for Copenhagen and Marseille) and used in the improved control logic. Certainly this is a simplified approach (the same weather data is assumed since we don´t have weather data for reference site, fixed PV efficiency, horizontal PV, same consumption power (heating), neglect different sunrise/-sunset time). But it demonstrates the universality of the system model and can be improved without much effort. The resulting simulations indicate e.g. that the maximum of H$_2$ mass in storage barely reaches 310 kg (9 % reduction) for Copenhagen, while for Marseille the maximum of H$_2$ mass reaches 366 kg (8 % increase). Further comparisons are displayed in Table 4.

3 Evaluation of results

3.1 Level of autarky

One of the key performance indicators for effectiveness of H$_2$ storages is the *Level of autarky* (*LoA*), indicating how self-sufficient the system is. We consider two levels of autarky here. One, in terms of energy taken from the grid LoA_{energy}, and the other with regard to time span of energy taken from the grid LoA_{time}. [4]

$$LoA_{energy} = 1 - \frac{\int_0^{1year} \left(P_{con}(t) - P_{prod}(t)\right)\sigma(t)dt}{\int_0^{1year} P_{con}(t)dt} \tag{10}$$

$$LoA_{time} = 1 - \frac{\int_0^{1year} \sigma(t)\; dt}{\int_0^{1year} t\; dt} \tag{11}$$

Here a time dependent characteristic function $\sigma(t)$ is introduced:

$$\sigma(t) = \begin{cases} 1 & if \text{ power taken from grid} \\ 0 & else \end{cases} \tag{12}$$

P_{prod} is the power actual provided by the internal system (PV, battery, FC) and P_{con} is the actual power demand. If $P_{prod} < P_{con}$ then the difference results in power taken from the grid. Comparison of different variants of H$_2$ storages with regard to level of autarky is summarized in Table 4.

Table 4. Comparison of different variants of H$_2$ storages with regard to level of autarky.

Type of H2 storage/capacity of storage [kg]	LoA_{energy}	LoA_{time}
Pressurized/400 (real)	0.91	0.93
Pressurized/300 (real)	0.89	0.91
Pressurized/200 (real)	0.87	0.89
LOHC/400 (real)	0.84	0.88
Pressurized/400 (ideal)	0.96	0.96
Pressurized/300 (ideal)	0.92	0.93
Pressurized/200 (ideal)	0.88	0.91
No storage (batteries only)	0.79	0.84
Pressurized/400 (realCopenhagen)	0.87	0.89
Pressurized/400 (realMarseille)	0.96	0.97

The results of ideal simulations, i.e. without physicalities show good agreement with the linear optimisation analysis (Table 2). However, one can see how the level of autarky drops when real behaviour of components is considered. The variant with 400 kg pressurized storage has the highest autarky level considering physicalities (LoA_{energy} = 0.91 and LoA_{time} = 0.93). Here the complex of households would take from the electric grid $\sim$ 3.4 MWh/year (out of total consumed el. power 37.5 MWh/year) and it would need to take electric power from the grid for $\sim$ 25 days out of the whole year. The worst combination regarding *LoA* is the LOHC variant due to the poor efficiency of the LOHC process (LoA_{energy} = 0.84 and LoA_{time} = 0.88). This variant would increase both LoA_{energy} and LoA_{time} by just 4% as compared to a system with batteries only, without H$_2$ storage. As one would intuitively expect, the difference in *LoA* is mainly driven by the time it takes until the H$_2$ storage is empty in autumn (Figure 6).

3.2 Cost Model

Together with the technical analysis, an economical analysis for 30 years of operation was performed as well. For this, a one year simulation is extrapolated to 30 years of operation by simple duplication. We are aware that this neglects possible future developments, fostering energy autarkic settlements. These are for example: increasing electricity price over time, additional profit from CO$_2$ certificate trade, control energy offer and negative electricity prices. Moreover governmental subsidiary programs for storage technologies as well as improved efficiency of FC/ELY and lower component prices. The simplified cost extrapolation can be seen as a conservative point of view: any of the mentioned future energy market and technol-

[4] Overproduced energy sent to the grid was not counted.

ogy improvements will lower our cost prognosis in favour of a storage technology. Determination of costs for the different H_2 storage variants was accomplished with help of the so called `collectCosts` models implemented in the TransiEnt Modelica library. They were included in the hydrogen system components and cover investment, operational and maintenance, demand (e.g. purchasing of el. energy), revenues (e.g. selling el. energy) and other costs. The economical analysis was performed under the following assumptions:

- Only hydrogen system components costs were considered (PV and batteries were the same for each variant so it was omitted). [5] (Table 5)
- Price of occupied land was included for storages only (300 €/m^2). (Table 5)
- Energy costs were included:
  Demand (from grid) as 0.3€/kWh
  Revenue (to grid) as 0.1€/kWh
- Lifetime of components was covered.
  ELY/FC : 10 years
  Bottles / Tank / Compressor: 20 years
  LOHC storage: 20 years
- Annuity factor for 30 years set for 1/30 (i.e. numerical zero interest rate).
- No operational and maintenance cost were assumed but is adaptable in the model [6] [7]

Table 5. Costs for hydrogen related components.

Components	Cost [k€]
ELY (2x5.8 kW units)	60
FC (2x4.7 kW units)	130
Bottles ({200/300/400} kg)	90/130/170
additional land cost	8/12/16
Tank ({200/300/400} kg)	220/360/440
additional land cost	17/26/34
Compressor	70
LOHC storage ({200/300/400} kg)	390/430/470
additional land cost	-/-/8

The final comparison of different variants of H_2 storages (type and capacity {400,300,200} kg) with regard to costs for 30 years of operation is summarized in Table 6. As expected, the cheapest variant for a H_2 storage is 200 kg with bottles. However, the 400 kg storage using bottles is "just" 150 k€ more expensive and offers much more flexibility, hence much higher level of autarky. The 400 kg storage using tanks is the most expensive variant (~1.5 times more than 400 kg bottle storage) even more expensive than the LOHC. At the moment the HB system is not economically competitive in comparison to standard energy supply. These findings are supported by Grosspietsch et al. (2018).

Table 6. Comparison of different variants of H_2 storages with regard to costs in k€ for 30 years of operation.

Storage/capacity	Invest.	Reven.	Deman.	Total
Bottles/400	956	50	34	941
Bottles/300	879	60	39	858
Bottles/200	813	81	55	786
Tanks/400	1365	50	34	1349
Tanks/300	1136	60	39	1115
Tanks/200	919	81	55	893
LOHC/400	1307	50	56	1313
No storage	0	147	72	-75

4 Summary & Outlook

From today's perspective, applying a H_2 storage does not pay off economically. This points towards a necessity for adjusting political/economical conditions as well as a need for technical improvements, see subsection 3.2. As for today, *pressurized* and *LOHC* storages of H_2 are the favourite choices in terms of technical readiness, safety and economy. Based on our results, a storage consisting of pressurized gas bottles gives the most economic solution. If the LOHC technology becomes more affordable in the future and most importantly its efficiency becomes higher or if there exists a possibility to recuperate the heat produced during hydrogenation then LOHC might become a real option. Safety is a pro of LOHC, since H_2 is stored under ambient condition and in a hardly inflammable state.

The developed system model including control logic is quite universal: without much modelling effort, it can be extended by other energy producers, e.g. by wind turbines and a wind forecast, while keeping the control logic model untouched. Also maintenance costs and annuities can be included. Hence, the energy system can be customized to specific technological set ups. Also the geographic location of the house complex can be varied by providing weather and modified (merely) heat consumption data. This enables location dependent design studies for residential complexes. Further, our model can be used for an optimised system design based on physicalities also regarding storage tank and battery capacity.

Our work demonstrates how a linear optimisation can guide the development of a more detailed physical Modelica model of the storage system. Simultaneously it points out the necessity of dynamic simulations in the design process of the storage: The effects of control decisions and

[5] Component cost data sourced: LOHC (Hydrogenious 2020), pressurized bottles (BBA 2017), pressurized tank (ELKUCH 2019).

[6] in (Gstöhl and Pfenninger 2020) 7.5 % of initial investment sum is assumed.

[7] The last two assumptions might seem too primitive. However there are other uncertainties, such as constant electricity price etc., which have larger impact on the costs. In the scope of this work it is sufficient for comparison of the technologies.

physicalities integrated over the investigated operational time period of 1 year have a substantial impact on the level of autarky (e.g. 96% → 91% wrt. total energy consumption) and maximum mass ($\sim$ 15% decrease) of stored hydrogen.

Modelica has proved to be an adequate numerical tool to tackle these kinds of analyses. Using a combination of TransiEnt and ClaRa library as well as e.g. XRG's HumanComfort and HVAC library opens the door for detailed system models of residential house complexes, including the buildings and their heating systems.

Acknowledgements

This work has been supported by the German Federal Ministry for Economic Affairs and Energy within the scope of the 6th energy research program (grant number 03ET6059A).

Supported by:
Federal Ministry for Economic Affairs and Energy
on the basis of a decision by the German Bundestag

The authors thank the anonymous referees for valuable comments and suggestions.

References

BBA (2017). Ed. by BBA Müller GmbH. fetched Apr., 6th 2021. URL: https://www.messe-essen-digitalmedia.de/uploads/E301/pdf/company/bba-mueller-gmbh-e2f6e-info.pdf.

Bensmann, B. et al. (2016). "Optimal configuration and pressure levels of electrolyzer plants in context of power-to-gas applications". In: *Applied Energy* 167.1–2, pp. 107–124.

Bentvelsen, R. F. P. (2019). "Modeling and Scheduling of a ControllableElectrolyser in an IndustrialGrid". MA thesis. Delft University of Technology, Netherlands.

BMU, ed. (2011). Berlin. URL: http://www.verwaltungsvorschriften-im-internet.de/bsvwvbund_13122011_KIIII54603022.htm (visited on 2021-04-11).

BMU (2021). *Gesetz für den Ausbau erneuerbarer Energien (Erneuerbare-Energien-Gesetz - EEG 2021)*. German. Ed. by German Federal Ministry of Justice and Consumer Protection and German Federal Office of Justice.

ClaRa v1.3.0 (2020). Ed. by ClaRa Development Team. fetched Apr., 6th 2021. URL: https://claralib.com/.

ELKUCH (2019). Ed. by LUDWIG ELKUCH AG. fetched Apr., 6th 2021. URL: https://www.elkuch.com.

Frey, Hartmut (2019). *Energieautarke Gebäude: Auf dem Weg zu Smart Energy Systems*. 1. Auflage 2019. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 3662578743.

Fritzson, Arash M.; Dizqah Alireza Maheri; Krishna Busawon; Peter (2013). "Modeling and Simulation of a Combined Solar and Wind Systems using OpenModelica". In: *Annual OpenModelica Workshop, LinkÃ¶ping University,Sweden*.

FuelCellStore (2020). Ed. by Fuel Cell Store. fetched Apr., 6th 2021. URL: https://www.fuelcellstore.com.

Grosspietsch, David et al. (2018). "How, When, and Where? Assessing Renewable Energy Self-Sufficiency at the Neighborhood Level". In: *Environmental science & technology* 52.4, pp. 2339–2348. DOI: 10.1021/acs.est.7b02686.

Gstöhl, Ursin and Stefan Pfenninger (2020). "Energy self-sufficient households with photovoltaics and electric vehicles are feasible in temperate climate". In: *PloS one* 15.3, e0227368.

Henriquez, A. M. (2018-06-08). "Model of hydrogen production system for investigating the energy flexibility of residential buildings". MA thesis. UNIVERSITY OF LIEGE BELGIUM, p. 76.

Hilpert, S. et al. (2018). "The Open Energy Modelling Framework (oemof) - A new approach to facilitate open science in energy system modelling". In: *Energy Strategy Reviews* 22, pp. 16–25.

HPS Home Power Solutions GmbH, ed. (2021). *HPS System – picea*. URL: https://www.homepowersolutions.de/produkt (visited on 2021-07-05).

HumanComfort v.2.11 (2020). Ed. by XRG Simulation. fetched Apr., 6th 2021. URL: https://www.xrg-simulation.de/en/products/xrg-library/humancomfort/.

HydrogenEurope (2020). Ed. by Hydrogen Europe AISBL. fetched Apr., 6th 2021. URL: https://www.hydrogeneurope.eu/.

Hydrogenious (2020). Ed. by Hydrogenious LOHC Technologies GmbH. fetched Apr., 6th 2021. URL: http://www.hystoc.eu/.

Kofman, G. Migoni; P. Rullo; F. Bergero; E. (2016). "Efficient Simulation of Hybrid Renewable EnergySystems". In: *International Journal of Hydrogen Energy* 41.32.

Kopp, M. et al. (2017). "Energiepark Mainz: Technical and economic analysis of the worldwide largest Power-to-Gas plant with PEM electrolysis". In: *International Journal of Hydrogen Energy* 42.19, pp. 13311–13320.

Krieger, Christoph (2019). "Process engineering consideration and optimization of the release of hydrogen from organic carrier materials (LOHC)". Doctoral dissertation. Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), p. 121.

Macagno, M. Santarelli; S. (2004). "A thermoeconomic analysis of a PV-hydrogen system feeding the energy requests of a residential building in an isolated valley of the Alps". In: *Energy Conversion and Management* 45 (3), pp. 427–451.

PHI (2021). *Passive House Institute (PHI) website*. URL: https://passivehouse.com/ (visited on 2021-04-13).

Proton Motor Fuel Cell GmbH (2021). *PEM-fuel cell*. URL: https://www.proton-motor.de/en/ (visited on 2021-04-21).

ResiliEnt.EE (2021). *- Resilience of integrated energy networks with a high share of renewable energies*. URL: https://www.tuhh.de/transient-ee/en/index.html.

Scarisbrick, Constantin (2019). "Simulation of a Fuel Cell and a Metal Hydride Storage System". Diploma Thesis. Technical University of Vienna, Austria. 73 pp.

Stiftung Umwelt Arena Schweiz, ed. (2021). *Umweltarena Brütten*. URL: http://www.umweltarena.ch/ (visited on 2021-07-05).

Tjarks, Geert et al. (2018). "Energetically-optimal PEM electrolyzer pressure in power-to-gas plants". In: *Applied Energy* 218.12, pp. 192–198.

TransiEnt v1.2.0 (2020). Ed. by TransiEnt Development Team. fetched Apr., 6th 2021. URL: https://www.tuhh.de/transient-ee/index.html.

University of Wisconsin–Madison Solar Energy Laboratory (1975). *TRNSYS, a transient simulation program*. Madison, Wis. : The Laboratory, 1975.

Vezzoli, Carlo (2018). *Designing sustainable energy for all: Sustainable product-service system design applied to distributed renewable energy*. Green energy and technology. Cham, Switzerland: Springer.

XRG Simulation (2021). "in-house Modelica library for Fuel-Cells and Electrolysers".

Modelling the Synchronisation Control for a Hydro Power Controller

Jonatan Hellborg Tonje Tollefsen Khemraj Bhusal Dietmar Winkler

Department of Electrical Engineering, IT and Cybernetics, University of South-Eastern Norway, Porsgrunn, Norway
(contact: dietmar.winkler@usn.no)

Abstract

This paper presents the modelling of a synchronisation control as used inside a typical hydro power controller for small hydro power plants. It was built using the open-source modelling language Modelica by use of the Modelica Standard Library (Modelica Association 2019), the OpenIPSL (ALSETLab et al. 2018) and the OpenHPL (TMCC 2019).

The resulting model allows for both transient and long-term simulations for the complete hydro power system with the main functions available and working. This includes water-level control, frequency control, voltage control with a power factor control and the synchronisation sequence.

Keywords: Modelica, hydro power, synchronisation, hydro power controller

1 Introduction

As the importance of the transition to a carbon-neutral society increases, so does the need for tools that accelerates this problem. Because Norway is a country with great possibilities for hydro power, both large and small, there exists an inherent need for tools that allows for accurate simulations of existing plants but also the ability to simulate plants before they are built.

Norway holds a special place in the European electrical power grid because of the large usage of hydro power. Not only is 98 % of Norway's electrical power hydro (Petroleum and Energy 2016), but it also accounts for an incredible 50 % of Europe's reservoir capacity. (Statkraft 2020)

As more focus is given to the transition to a completely carbon-neutral society, the development of small scale hydro power has seen an increase, and with this increase comes the need for better tools that allow for better and more accurate simulations. These tools and the knowledge required to properly utilise them will help engineers and designers to find potential design flaws earlier in the design process, before the turbine is deployed. This becomes more and more important as more smaller companies (or even municipalities) decide on building small hydro power systems.

2 The Hydro Power Controller

A typical hydro power controller is made up of several components that realise different key functionalities of a hydro power station. They can be grouped into two main groups:

- The turbine governing functions act on the hydro turbine by providing the control signal for the guide vanes or nozzle opening (depending on the turbine type).
- The generator governing functions provide the signal for the excitation system in order to control the voltage and/or power-factor.

The controller implemented as part of this work is a typical hydro power controller for small hydro power stations (with a power rating between $1\,MW$ and $10\,MW$). It provides the following functionalities:

1. Turbine governing
   (a) Speed control
   (b) Water-level control
2. Generator governing
   (a) Voltage regulation
   (b) Power-factor control

A typical hydro power controller contains several additional functions which are not focus of this work.

2.1 Turbine Governing

The Turbine governor is the main controller of the hydraulic turbine. The governor regulates the turbine-generator speed by controlling the flow rate of the water into the turbine. A higher water flow through the turbine means more torque is applied to the turbine runner which results in a higher applied mechanical power. Depending on the power balance the speed of the turbine and the directly connected generator will either increase or decrease.

The system of governing basically consists of a *control section* and a (mechanic/hydraulic) *actuation section*, see Figure 1.

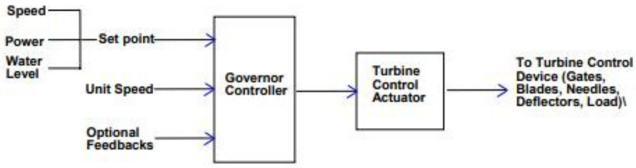


Figure 1. Basic Governor Control System (OD Thapar 2008)

2.1.1 Speed Control

The speed control is responsible for keeping the speed at which the turbine is spinning as close as possible to a given set-point so that the generator, which is directly coupled with the turbine, can produce the correct frequency. The controller has four different operation modes:

- Disconnected
- Normal
- Unstable
- Isolated

Each of these modes uses different sets of control parameters to suite the different nature of the electrical load experienced by the generator.

2.1.2 Water-Level Control

The main purpose of the water-level control in the hydro power system is to maintain an optimal water-level at the intake. This kind of control is especially common for small hydro power stations with small intake reservoirs (e.g., run-of-river types) where the varying precipitation can cause quite a change in the water-level. Operators use different strategies for their hydro power plants which means that the water-level controller needs to accommodate a different water-flow to guide vane opening characteristic.

A PI controller usually controls the output signal for the guide vanes for bringing the process variable (i.e., the water-level) to the desired value. If there is increment of the water-level at the intake, the guide vanes opening of the turbine should then respond by increasing the opening of the guide vanes so that the desired set point is reached.

If the water-level is below any specific level, the water-level controller will stop the turbine. This is important to avoid running out of water which then would lead to air getting into the waterway.

2.2 Generator Governing

2.2.1 Voltage Regulation

When a varying load is subjected to a generator then voltage at the armature terminals will vary to a certain extent. The amount of this variation determines the regulation of the machine. The voltage regulation,VR, of a generator can be defined as the change in terminal voltage from full load, V_{fl}, to no load, V_{nl}, expressed as a percentage of full load volts, when the speed and excitation field current are held constant (Pterra Consulting 2021).

$$VR = \frac{V_{nl} - V_{fl}}{V_{fl}} \cdot 100\% \tag{1}$$

The full-load voltage is the terminal voltage when full load current is drawn. The no-load voltage is the one when zero current is drawn from the supply, i.e., open circuit terminal voltage.

The purpose of a voltage regulator is to keep the voltage within the prescribed range.

The voltage regulation is normally achieved by an *excitation system* which usually consists of an automatic voltage regulator (AVR), an exciter, measuring elements, a power system stabiliser (PSS) and limitation and protection units, see Figure 2.

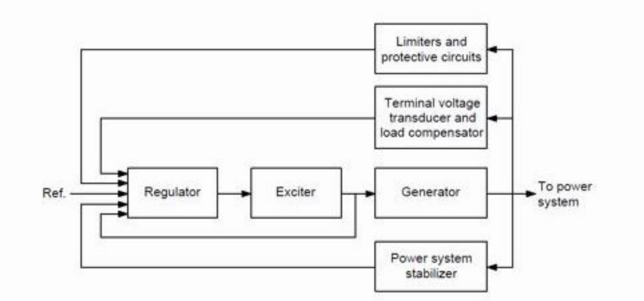

Limiters and protective circuits
Terminal voltage transducer and load compensator
Ref.
Regulator
Exciter
Generator
To power system
Power system stabilizer

Figure 2. Block diagram of an excitation system of a synchronous generator (Pterra Consulting 2021)

The current and voltage produced by the exciter is usually controlled by the AVR. In case of sudden disturbances, there may be negative influences on the damping of power swings. For this, a supplementary control loop, the PSS, is introduced. The PSS produces an additional signal which is injected into control loop in order to compensate any voltage oscillations.

2.2.2 Power Factor Regulator

In general, an excitation system used in synchronous generators is expected to aid in the regulation of the system grid voltage. However, for small hydro power plants, the voltage regulation from the excitation system will not have a large impact on the system grid voltage. Therefore, operators of small hydro power plants often utilise a power factor regulator to regulate the excitation voltage so that the generator operates at power factor 1. The power factor regulator is connected to the AVR as an outer loop control that takes direct control of the excitation voltage. It is important to note that a power factor regulator may not be suitable to handle voltage instability that may occur from faults, and therefore should not act as a replacement for the AVR entirely (IEEE 2016).

2.3 Synchronisation

Synchronisation is the process of synchronising the generator terminal voltage(s) with an existing distribution grid in order to make an electric connection. This process should be executed as such that the least amount of disturbance is caused to both, the generator and the grid.

The generator synchronisation conditions are as follows (Chapman 2012):

1. The RMS line voltages of the generator must be equal to the electrical grid. Otherwise, it may cause a large current flow when the switch is closed.

2. The generator must have the same phase sequence as the grid; in order to avoid different current flow between the phases.

3. The phase angles of the generator and the electrical grid phases must be equal.

4. The generator frequency must be slightly higher than the nominal frequency of the electrical grid. This is to prevent the generator from consuming power instead of supplying power, during frequency stabilisation. If the frequencies are not nearly equal, it will cause large power transients to the generator during the connection process until it stabilises at a common frequency.

In order to achieve the synchronisation process successfully a hydro power controller needs to engage the turbine governing and generator governing functions in a well planned sequence.

3 Modelling

All the main controller functions as described in the previous section are implemented in Modelica (Modelica Association 2013) by use of components from the Modelica Standard library (Modelica Association 2019).

The controller model is split up into separate submodels for each of the sub-controllers function:

- Synchronisation Control

- Frequency Control

- Water-level Control

- Excitation System Controller

Those sub-models are then combined to make up the complete hydro power controller as shown in Figure 3.

Each of these sub-controllers are active during different conditions and during different steps in the synchronisation sequence. An example of this is the frequency controller which is only active before synchronisation, with the objective of bringing the generator up to its nominal speed so that the frequency matches that of the grid. After this has been achieved, the frequency controller is deactivated and the water-level controller is activated which ensures maximum power is produced.

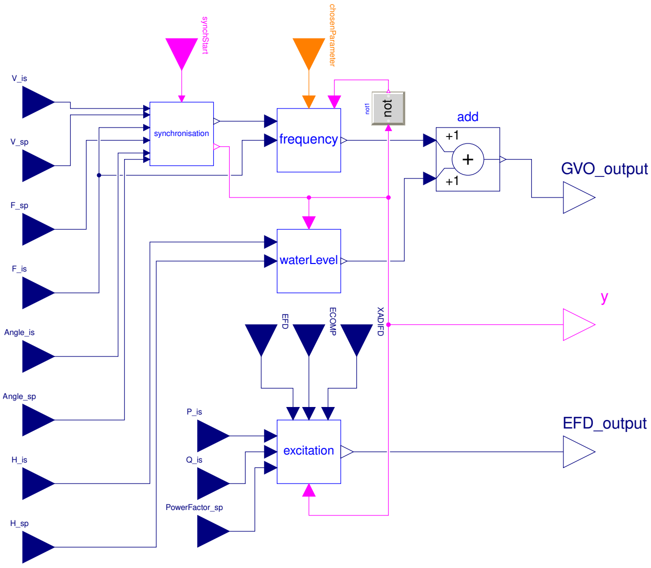


Figure 3. Implementation of the hydro power controller in Modelica

3.1 Synchronisation Control

The model of the synchronisation control is shown in Figure 4.

The synchronisation control decides when to close the circuit breaker and to do this it takes in 6 measurement inputs (V_{grid}, V_{gen}, f_{grid}, f_{gen}, $\text{angle}_{\text{grid}}$ and $\text{angle}_{\text{gen}}$) as well as a `true`/`false` signal that tells the controller when to allow synchronisation. The controller checks if the measurements are within a set range of each other and if they are, the corresponding block sends a `true` signal which activates the next block. The sequence of checks are as follows:

1. Voltage

2. Frequency

3. Angle

4. Phase sequence

Normally, the synchronisation would only be permitted when all of the above measurements are within the allowed range. However, in the developed model the last requirement (phase sequence) is ignored because it is assumed to be the same for the generator and the grid at all times.

The synchronisation controller will also increase the frequency set-point with a configured offset. This is to ensure that the generator operates at a slightly higher speed than the grid. The reason behind this is that when the grid and generator synchronises, the generator (operating at a higher speed) will be slowed down causing a sudden burst of energy to flow into the grid. This is preferable to the alternative where the grid has a slightly higher frequency than the generator, as this would cause the generator to act

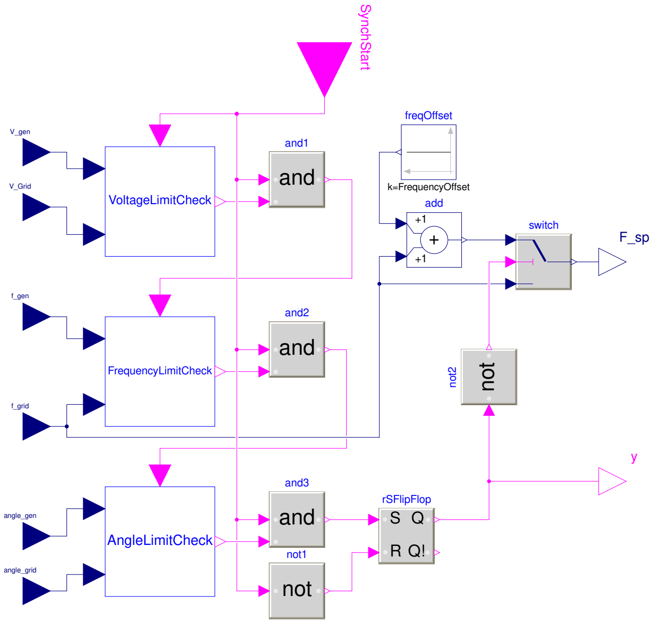


Figure 4. Synchronisation controller in Modelica

like a motor for a short moment. It is important to note that this difference should be kept at a minimum, most preferable zero, but that a slightly higher generator frequency is better than a lower one (Chapman 2012).

3.2 Frequency Control

The frequency controller (often also called speed controller) is based on a PID Structure with some added functionality, see Figure 5. The frequency controller has four different sets of PID parameters, each active during different operational modes:

- Disconnected
- Normal
- Unstable
- Isolated

These four parameter sets are implemented in Modelica using four different sets of component, giving four different outputs. Depending on the given integer input the correct output is selected and is used as the output for the frequency controller.

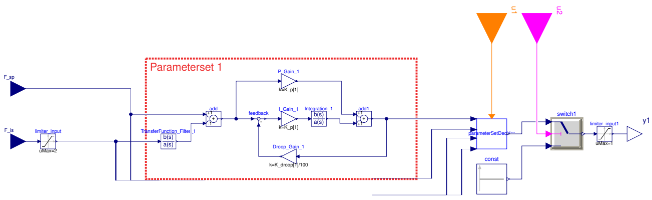


Figure 5. Frequency controller in Modelica

3.3 Water-level Control

The water-level controller implemented in Modelica is shown in Figure 6. The purpose of this controller is to allow the hydro power plant to produce as much electrical power as possible when there is enough water. This means that if the water-level falls below a certain set-point, defined in the controller, then the controller will decrease the guide-vane opening of the turbine and thus the mechanical power. One of the requirements of this controller is that the generator is already synchronised to the grid as the frequency needs to be stabilised before the water-level controller is active.

The characteristic of the guide vane opening with respect to the water-level can be adjusted via the look-up table `WaterLevelLinearization`. Depending on the operator the strategy of how much power (i.e., guide vane opening) should be produced can be different. One strategy can be that one runs the turbine with full guide vane opening as soon as the water-level is above the medium set-point. Others might prefer a more linear dependency as shown in Figure 7.

3.4 Excitation System Controller

The excitation system controller can consist of many functions that will affect the voltage reference, such as (IEEE 2016):

- Power system stabilizer (PSS)
- Reactive compensation
- Active compensation
- Frequency droop
- Limiters
  - OEL - Overexcitation Limiter
  - UEL - Underexcitation Limiter
  - SCL - Stator current limiter
- VAR regulator and PF regulator

The purpose of the excitation system controller is to regulate the terminal voltage to different levels that will satisfy various generator operations and system stability conditions. The excitation system controller used in this work is based on the standard static excitation system model *ST1A* from *IEEE 412.5* (IEEE 2016). The *ST1A* is a potential-source controlled-rectifier excitation system. The excitation system is getting its excitation power supplied through a transformer that is connected to the generator terminals. It contains a PI voltage regulator that compares the generator terminal voltage against the voltage reference, a forward path transient gain reduction, and a feedback loop which acts a stabilising effect on the the excitation.

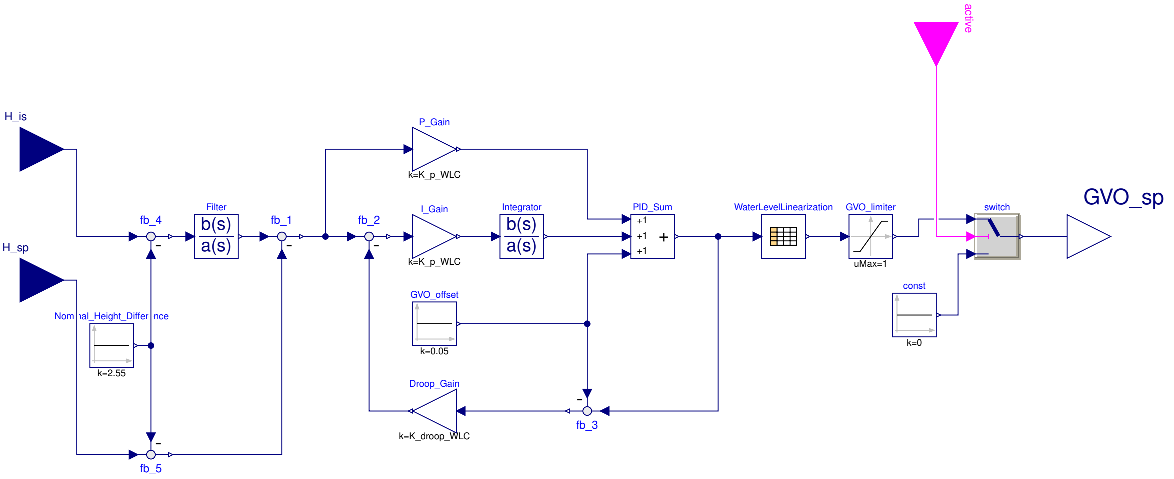


Figure 6. Final implementation of the water-level controller in Modelica

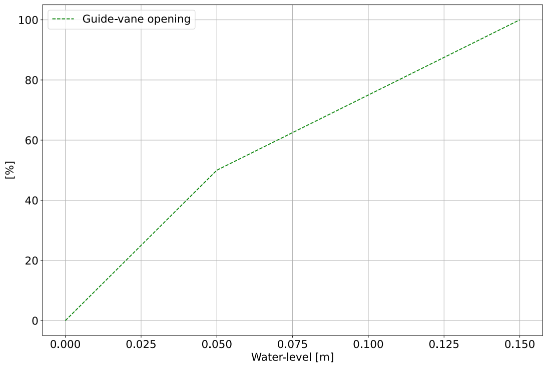


Figure 7. Non-linearity of water-level

As shown in Figure 8 the block diagram of the ST1A model has several inputs like the generator terminal voltage V_C, the voltage reference V_{ref}, the overexcitation limiter V_{OEL}, the generator field current I_{FD}, and alternative input points for the Power system stabiliser V_S and the underexcitation limiter V_{UEL} (IEEE 2016).

This paper is limited to only include power factor regulator function. Therefore the limiters and stabilisers are disconnected.

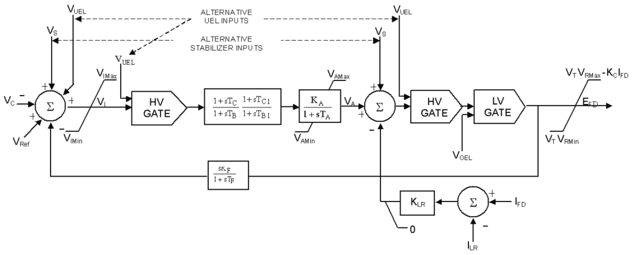


Figure 8. Block diagram of excitation system ST1A from IEEE 421.5-2005 (IEEE 2016)

The *ST1A* excitation system controller is available in the OpenIPSL (ALSETLab et al. 2018) as a PSS®E verified model. A block diagram of the developed excitation system is shown in Figure 9.

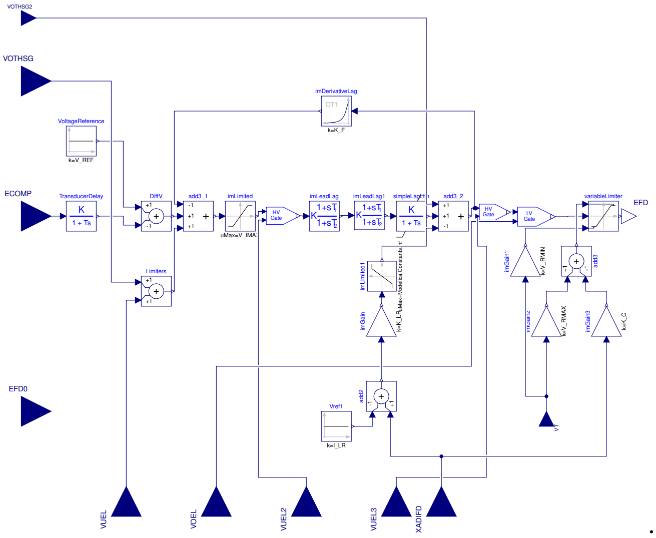


Figure 9. Final implementation of the excitation system ST1A developed in OpenIPSL

3.4.1 Power Factor Regulator

The power factor regulator is modelled similar to a voltage regulator, but the voltage input is replaced with a active power P input and reactive power Q input. The power factor regulator shown in Figure 10 is a simplified model based on the *standard power factor controller type 2* from IEEE 421.5 (IEEE 2016).

The power factor regulator consists of a power factor normalizer and a PI regulator. It controls the excitation voltage based on whether the generator is operating with a leading power factor or a lagging power factor, i.e., the generator is operating in the overexcited or underexcited region, respectively.

The relationship between variation of excitation field and power factor are displayed in Figure 11 (a). In order for the power factor regulator to recognise whether the generator is operating in overexcited- or underexcited region, and to deliver the proper control action, the power factor curve needs to be modified to a normalised definition of the power factor. This can be achieved with the following approach shown in Figure 11 (c). Also, this approach will ensure that the normalised power factor is a continuous function as shown in Figure 11 (b).

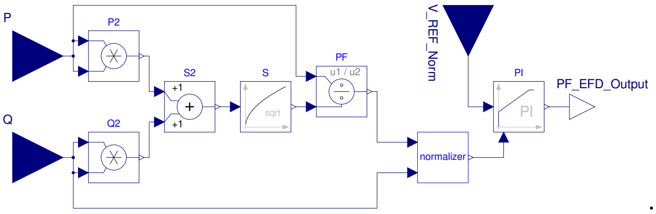


Figure 10. Model of *pf controller type 2*(IEEE 2016)

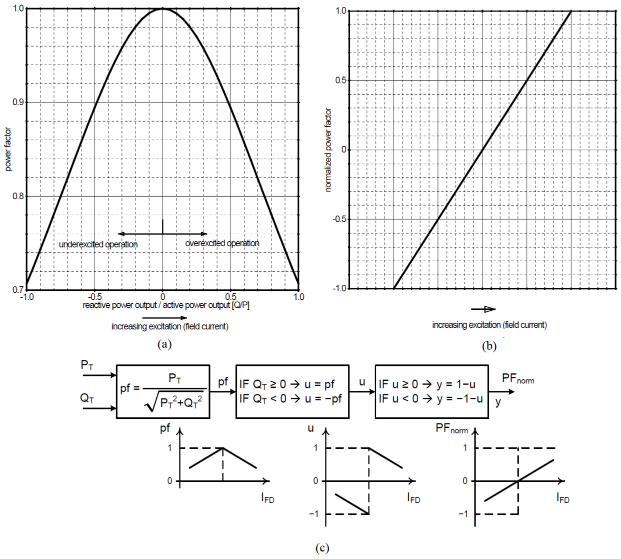


Figure 11. (a) power factor of a generator curve with varying excitation field, (b) normalised power factor from -1 to +1, (c) model logic scheme for normalised power factor from -1 to +1 (IEEE 2016)

4 Simulation Results

4.1 The Test-Bed model

The main test-bed model used in order to simulate and verify the implemented model is shown in Figure 12.

It contains the generator model `GENSAL` from the OpenIPSL, a small simplified grid with an infinite bus to connect to, the hydro power controller as described in section 3 and a model of a real waterway.

The waterway used is a model of a typical small hydro power system in Norway and was created using OpenHPL (TMCC 2019) and is shown in Figure 13. The hydro power plant is run-of-the-river type, where no upstream impoundment is built for water storage. The usable gross-head is about 60 m water column. The plant consists of a 2 MW Francis turbine connected to a 2.2 MVA synchronous generator.

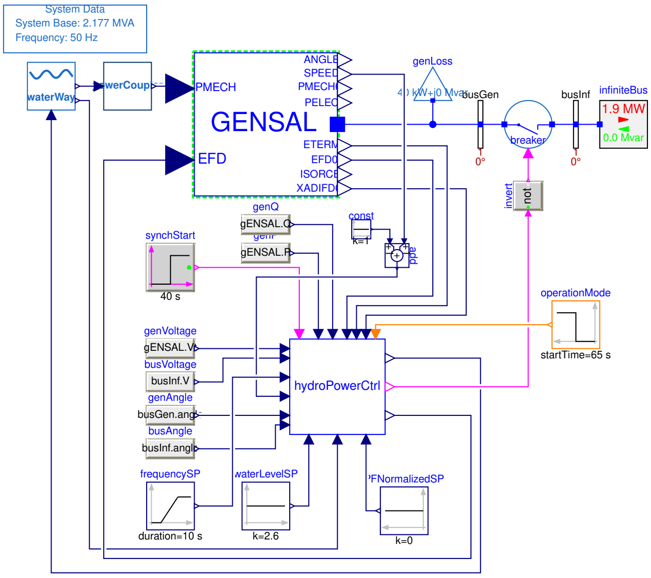


Figure 12. Test-bed model in Modelica

The test-bed model is using real water-level time series data stored in CSV format which then was read directly using the `ModelicaTableAdditions` library (Beutlich and Winkler 2021).

4.2 Synchronisation

The first action that the hydro power controller will take is to bring the generator from standstill up to the nominal speed. In order to simulate the gradual ramp-up that would occur in the real power-plant, the frequency set-point is changed using a ramp, which starts ramping up at 10 seconds. The result is a nice and smooth acceleration up to the nominal set-point, which can be seen in Figure 14.

The large step in the guide-vane opening comes from the fact that the water-level control will take over after the synchronisation in order to produce the maximum power possible. The generator frequency is locked to the infinite bus. Therefore an increase in guide-vane opening will not affect the speed of the generator, but will result in an increase in generator active power output.

During synchronisation the generator will try to reach a slightly higher speed than the grid is operating at which is the reason for the peak in active power at around 50 seconds. This is because when the grid and generator is synchronised, the generator will decelerate and therefore deliver a peak of power to the grid.

The turbine is accelerating up to the nominal speed, the voltage controller will control the excitation voltage of the generator and will adjust the terminal voltage of the generator to be close to equal the voltage of the grid as seen in Figure 15. After the synchronisation process is completed and the circuit breaker is closed, the generator terminal voltage will be locked to the grid, and the power factor controller will become active, aiming to keep the

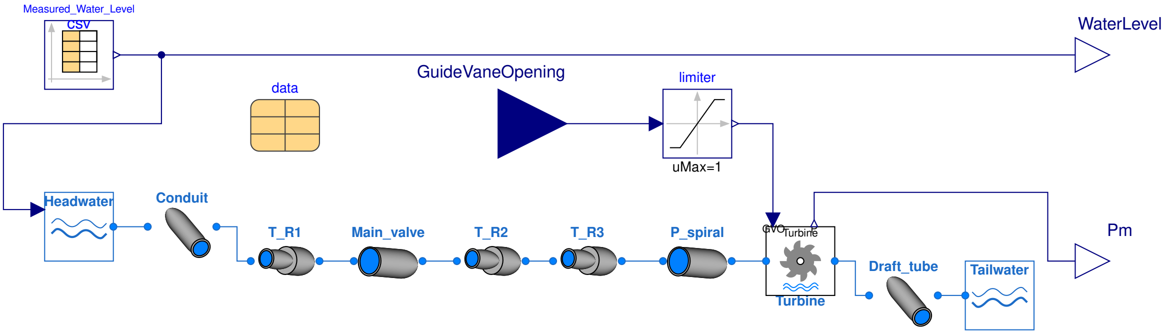


Figure 13. Model of the water way using OpenHPL (TMCC 2019)

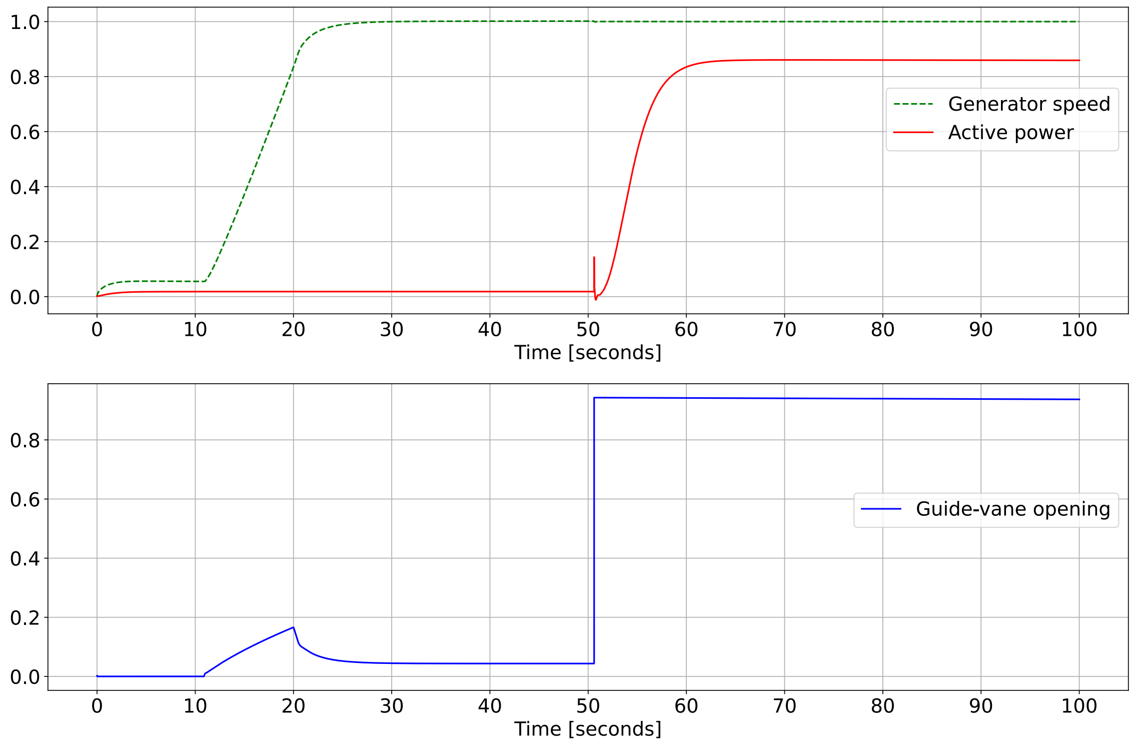


Figure 14. Frequency controller during synchronisation

generated reactive power close to zero.

When the circuit breaker closes at around $t = 52\,sec$, there is a slight reactive power peak that is being pushed to the grid. This is because of the generator terminal voltage being slightly higher, and not exactly equal to the grid. The power factor controller reacts to this reactive power peak with a dip in the excitation field (EFD) output.

When the active power production of the generator increases, the generator will start to consume a small amount of reactive power as a response. Therefore, we can see in Figure 15 that the voltage controller due to the power factor controller's influence, will increase the excitation output to compensate for the active power increase, and thus regulate the reactive power to close to zero, making the generator operate at power factor 1. Considering that the terminal voltage is locked to the grid, and the active power is fixed to mechanical power, an EFD increase will only affect the reactive power.

As mentioned in section 3, there are three conditions before synchronisation is allowed. These are:

- Voltage within limits
- Frequency within limits
- Angle within limits

Before the synchronisation is completed, the frequency set-point is set a bit higher than that of the grid. This offset between the grid and the generator is set to $0.02\,pu$ which is close to $1\,Hz$, and when the generator frequency is within the chosen limits, a `true` signal is given, activating the block that compares the generator angle to the grid angle. This frequency behaviour can be seen in Figure 16 and the angle behaviour in Figure 17.

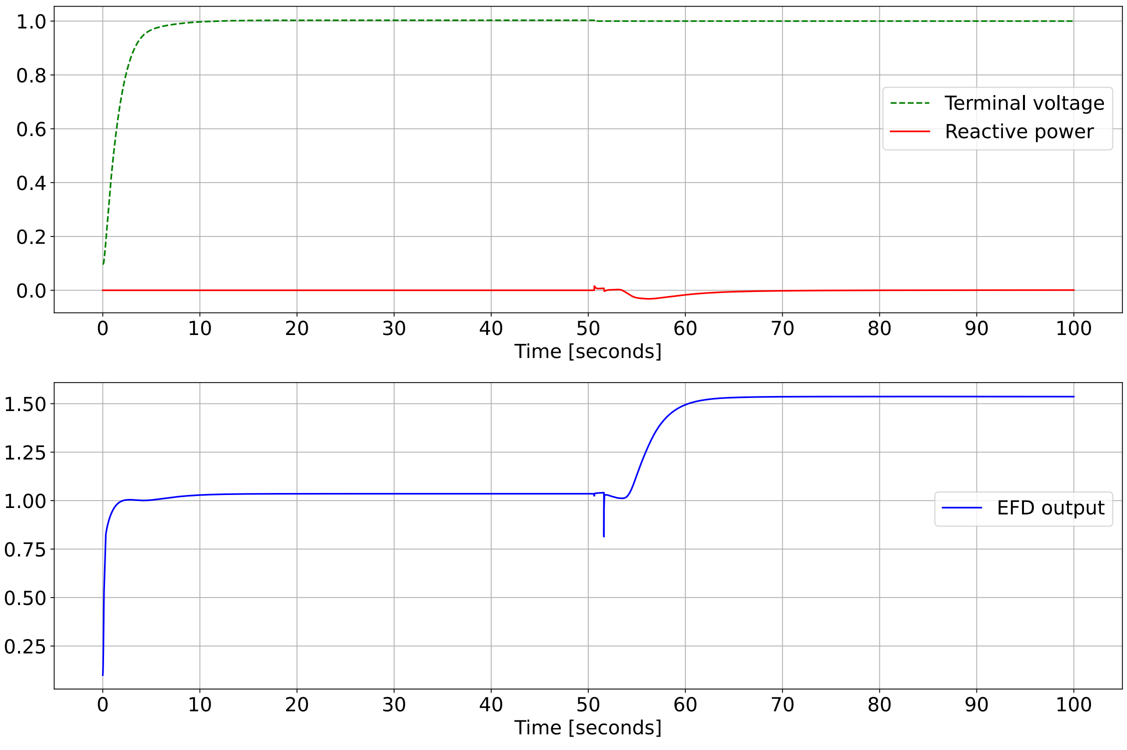


Figure 15. Voltage controller during synchronisation

During the synchronisation it is important that the voltage angle of the generator is leading the bus angle. Otherwise the generator will start as a motor and start consuming power.

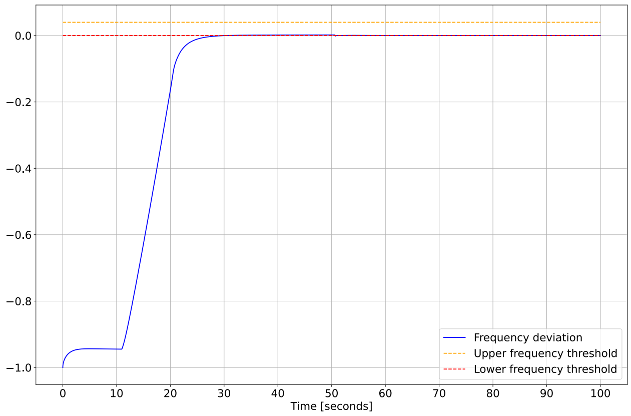


Figure 16. Frequency with the chosen limits

The resulting Boolean sequence of the synchronisation process is shown in Figure 18. Here a `true` signal is presented on “initiate synchronisation” only when each of the three variables are within the chosen limits (the angle is not shown). During synchronisation when the generator is slowed from the slightly higher frequency down to the nominal frequency of the grid, the frequency will dip slightly below the limits and will thus be `false` for a fraction of a second. This behaviour is what can be seen in the middle graph of Figure 18 but as soon as the last check turns `true`, the circuit breaker is locked so that a sudden frequency dip will not cause the system to fall out of synchronism again.

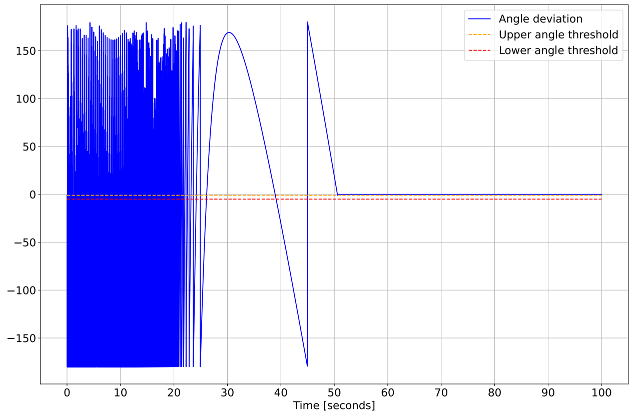


Figure 17. Angle with the chosen limits

5 Discussion

One of the assumptions made during the modelling of the synchronisation sequence is that the phase sequence for the generator is the same as for the grid. While this is not necessarily true, it is an reasonable assumption. Because of the simplicity of the `InfiniteBus` component used to simulate the overarching grid, which provides no means of measuring the phase sequence, adding the functionality

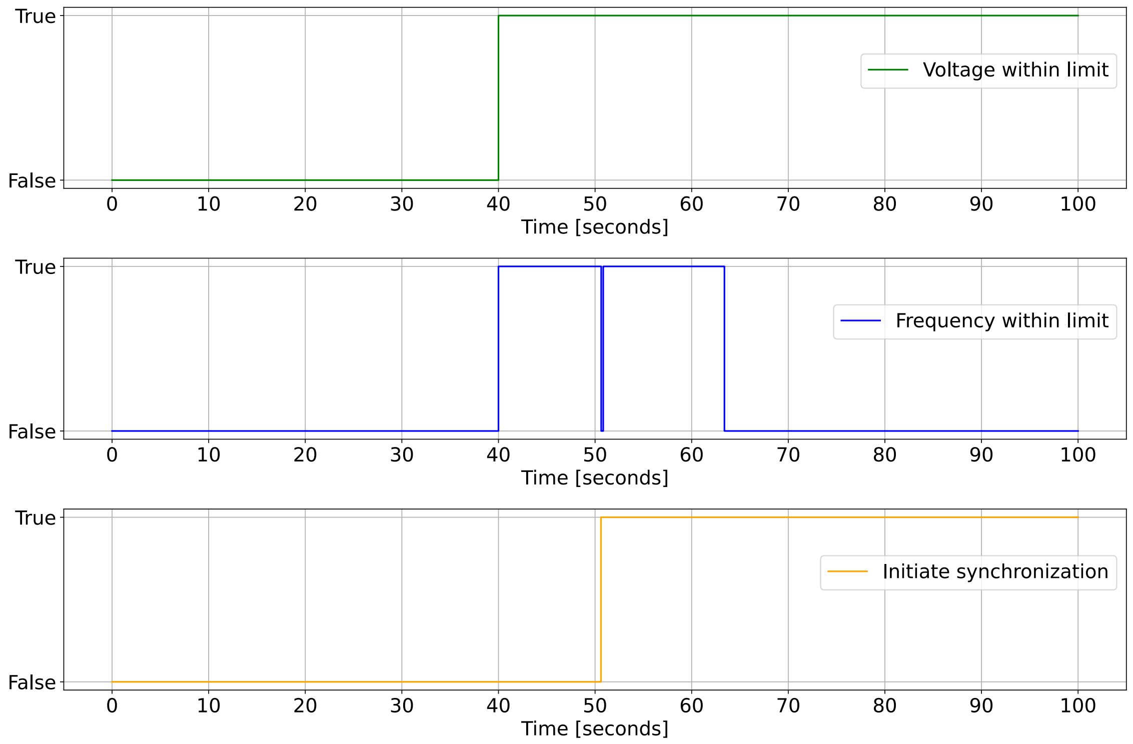


Figure 18. Synchronisation sequence

to the controller would be pointless as there is no way to verify it.

One of the bigger problems during the project was when it comes to verification is that for a lot of these values there was no real measurements available which means that these could not be verified. However, because most of these values still behaved realistically it was decided that this was good enough but that one of the goal with any future work is to attempt to get access to more measurements and verify the simulated values.

While some documentation regarding the different functions of the controllers was available som parameters had to be estimated. This means that for some of the functions, mainly voltage controller, a simulated version of the real controller could not be achieved as not enough information on the exact internal design was available.

6 Conclusions

Based on the documentation and parameter values received from power system operators and manufacturers, a model of the turbine controller was created using the Modelica Standard Library, the OpenIPSL and the OpenHPL.

During modelling, the finalised controller was divided up in to several sub-controllers. These were modelled individually and then fitted together into the finished model. The main controllers developed are the controllers for the water-level, frequency and power factor and these were then compared to the theoretical behaviour of these types of controller.

Overall the model functions and behaves as desired, but still needs proper tuning and further development.

7 Further Work

Some of functions typically present in a hydro power controller and outlined in section 2 are still missing. Future projects will continue to improve the model and also gather more data for verification. This also includes the grid model that will be replaced with a more complex model including station transformers and distribution grid components.

A continuation of this project has already been successfully completed and the results can be found in the paper “Developing Protective Limiters for a Hydro Power Controller in Modelica” (Manoranjan and Winkler 2021).

References

ALSETLab et al. (2018). “OpenIPSL: Open-Instance Power System Library – Update 1.5 to "iTesla Power Systems Library (iPSL): A Modelica Library for Phasor Time-Domain Simulations"”. In: *SoftwareX* 7, pp. 34–36. ISSN: 23527110. DOI: 10.1016/j.softx.2018.01.002.

Beutlich, Thomas and Dietmar Winkler (2021). “Efficient Parameterization of Modelica Models”. In: *Proceedings of the 14th International Modelica Conference*. 14th International Modelica Conference. Linköping Electronic Conference Proceedings. Linköping , Sweden: Linköping University Electronic Press, Linköpings universitet. URL: https://github.com/tbeu/ModelicaTableAdditions.

Chapman, Stephen J. (2012). *Electric Machinery Fundamentals*. 5th ed. New York: McGraw-Hill Higher Education. ISBN: 978-0-07-132581-3.

IEEE (2016). *421.5-2016 - IEEE Recommended Practice for Excitation System Models for Power System Stability Studies*. IEEE. DOI: 10.1109/IEEESTD.2016.7553421.

Manoranjan, Luxshan and Dietmar Winkler (2021). "Developing Protective Limiters for a Hydro Power Controller in Modelica". In: *Proceedings of the 14th International Modelica Conference*. 14th International Modelica Conference. Linköping Electronic Conference Proceedings. Linköping , Sweden: Linköping University Electronic Press, Linköpings universitet.

Modelica Association (2013). *Modelica – a Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.2 Revision 2*. Linköping: Modelica Association. URL: https://www.modelica.org/documents/ModelicaSpec32Revision2.pdf.

Modelica Association (2019). *Modelica Standard Library*. Version 3.2.3. URL: https://github.com/modelica/ModelicaStandardLibrary.

OD Thapar (2008). *Modern Hydroelectric Engineering Practice in India: Electro-Mechanical Works*. Vol. 1. Department of Hydro and Renewable Energy. URL: https://www.iitr.ac.in/departments/HRE/uploads/modern_hydroelectric_engg/vol_1/Chapter-6_Hydro-Turbine_Governing_System.pdf.

Petroleum, Ministry of and Energy (2016). *Renewable Energy Production in Norway*. URL: https://www.regjeringen.no/en/topics/energy/renewable-energy/renewable-energy-production-in-norway/id2343462/ (visited on 2020-08-31).

Pterra Consulting (2021). *Approaches to Complying with NERC Standard PRC-019-2 on the "Coordination of Generating Unit or Plant Capabilities, Voltage Regulating Controls, and Protection"*. URL: https://www.pterraph.com/nerc-compliance/approaches-to-complying-with-nerc-standard-prc-019-2-on-the-coordination-of-generating-unit-or-plant-capabilities-voltage-regulating-controls-and-protection/ (visited on 2020-11-16).

Statkraft (2020). *Hydropower*. URL: https://www.statkraft.com/what-we-do/hydropower/ (visited on 2020-08-31).

TMCC (2019). *OpenHPL: Open Hydro Power Library*. Version 1.0.0. University of South-Eastern Norway. URL: https://openhpl.simulati.no.

Developing Protective Limiters for a Hydro Power Controller in Modelica

Luxshan Manoranjan[1] Dietmar Winkler[2]

[1]University of South-Eastern Norway, luxshan@hotmail.no
[2]Department of Electrical Engineering, IT and Cybernetics, University of South-Eastern Norway, Porsgrunn, Norway,
dietmar.winkler@usn.no

Abstract

In recent years the operation of electrical power plants has become more and more challenging due to a more dynamic operation pattern in order to keep the voltage quality within the limits of what the electrical network regulators allow. This is due to the ever increasing amount of unregulated renewable energy (e.g., wind, solar, tidal power). There is a need for better tools that allow for a better and more accurate simulation of the operation of a electrical power plant. This paper presents the development of protective limiters as used in a typical hydro power controller. The limiters have been implemented using the Modelica language (Modelica Association 2017) and are according to the IEEE Std 421.5-201 (IEEE 2016). Having the limiters available in Modelica makes it possible to integrate them with hydro power system models build with the use of OpenHPL (TMCC 2019). The behaviour of the limiters have been tested against a verified generator model of the OpenIPSL (ALSETLab et al. 2018) comparing the theoretical behaviour.

Keywords: hydro power, Modelica, excitation system, protective controller, limiter

1 Introduction

The electrical power demand is still increasing, and it leads to pushing the society to find a renewable source to produce electricity. Therefore the development of existing and new hydropower stations is still increasing. The development of hydropower plants focuses not only on larger hydropower plants but also on small-scale hydropower plants in order to utilise as much resource from nature.

A hydropower plant consists of several components such as a valve, turbine, generator, etc. And one of such important components is the generator that converts mechanical energy to electrical energy. A generator needs an excitation system to provide field current to the field winding in order to induce the voltage in the generator terminal. An excitation system contains mainly an exciter that produces field current and an excitation control system that consists of an Automatic Voltage Regulator (AVR), controllers, and protective limiters to operate the generator and exciter within their capability in order to prevent destruction.

An AVR mainly controls field voltage, thereby the field current in order to obtain the desired output concerning the reference. Whereas, the controllers influence the AVR's reference to obtain the desired output, such as terminal voltage, power factor, or reactive power. The limiters influence the AVR to protect the generator by limiting the field and stator current to prevent the overheating of the field and stator winding, loss of synchronisms, and loss of excitation relays.

There are different type of conventional and specialised protective limiter has been used all over the world. This paper focuses on developing the conventional limiters to study their behaviour.

2 Theory

In order to understand the workings of the implemented limiters some basic theory knowledge is required. The following background information is mainly based on (Kundur, Balu, and Lauby 1994) and (IEEE 2016). A synchronous generator must be operated within its limits for active and reactive power output in order to not exceed the thermal capability of different components. The limiters ensure that exciters and synchronous generators are not exceeding their capability limits during normal and abnormal operating conditions. Therefore, the limiters comprise several types of control and protective functions. Most common limiters are determined using the generator capability curve (GCC) of a specific generator.

Figure 1 depicts the GCC of a synchronous generator, where curve A is a Field Current Overexcitation Limiter (FCOEL), also called Overexcitation Limiter (OEL), that limits the field current during the overexcited (exporting reactive power to the grid) operation. And curve B is a Stator Current Underexitation Limiter (SCUEL), also called Underexcitation Limiter (UEL), that prevents the excitation level from falling below the limit concerning active and reactive power or current during the underexcited (importing reactive power from the grid) operation. Curves C represents stator winding limits that are protected by the Stator Current Limiter (SCL). The Volts-per-hertz limiter is another limiter being used to protect equipment in the power plant.

DOI
10.3384/ecp21181617

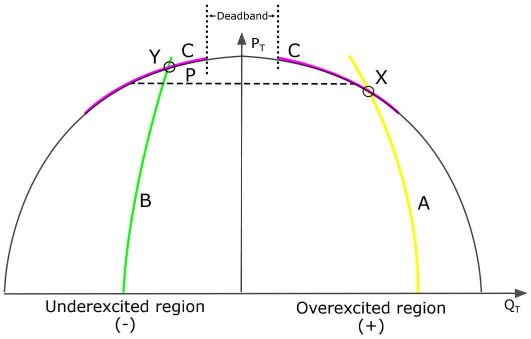


Figure 1. Synchronous generator capability curve with standard limits A: Field current limits, B: Stator end region limit and, C: Stator winding limit (between X and Y), and P: Turbine power (IEEE 2016)

2.1 Field Current Overexcitation Limiter (FCOEL)

The field current overexcitation limiter protects the generator from overheating due to prolonged field overcurrent. Simultaneously, it allows the maximum field forcing for power system stability purposes. The generator field winding is designed to operate continuously at rated load conditions. But during voltage collapse or system islanding, the power system will be stressed and cause the generator to operate at high levels of excitation for a period. This limiter measures the field current, field voltage, or exciter field current or voltage to detects overexcitation. When the overexcitation is detected, it allows continuing the overexcitation for a certain period, defined as the time-overload period, and then reduce the excitation level to a safe level. If this function does not reduce the excitation to a safe value, the FCOEL limiter will trip the exciter field breaker.

The FCOELs have two types of time-overload periods that allow overexcitation, inverse time or fixed time. The inverse time limiters operate with the time delay matching the generator's field thermal capability, as shown in Figure 2. While the fixed time limiters operate when the field current exceeds the pickup value for a fixed set time, irrespective of the degree of overexcitation. Currently, a more common type of FCOEL is a combination of both instantaneous and inverse-time pickup characteristics.

2.2 Stator Current Underexcitation Limiter (SCUEL)

The Stator current underexcitation limiter prevents the reduction of the excitation level of the synchronous generator by increasing the excitation in the generator for one or more following purposes:

- To prevent operating beyond the small-signal (steady-state) stability limit of the synchronous generator, which could lead to loss of synchronism.
- To prevent loss-of-excitation relays from operating during underexcited operation.
- To prevent overheating in the stator end region of the synchronous generator, typically defined by GCC.

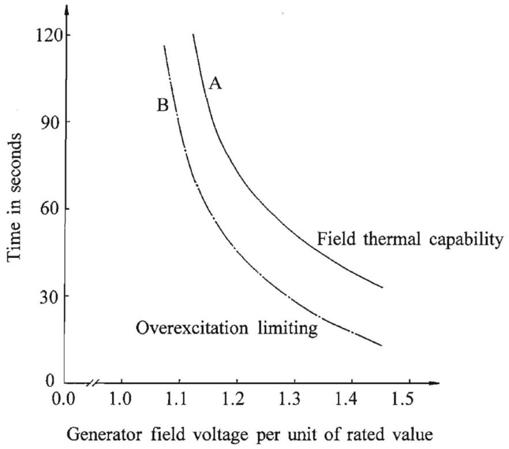


Figure 2. Coordination of overexcitation limiting with field thermal capability (Kundur, Balu, and Lauby 1994)

The SCUEL typically uses a combination of either voltage and current or active and reactive power of the synchronous generator to determine the control signal. Most importantly, the limiter should be coordinated with the required protection purposes as mentioned above in order to protect the generator properly. Figure 3 demonstrates a coordination of the calculated small-signal stability limit (I) and loss- of-excitation relay characteristic (II), where the intention was to protect against small-signal stability (I). If the UEL is supposed to protect against overheating in the stator end region, the coordination will be the same, but the small-signal stability limit is replaced by the overheating limit.

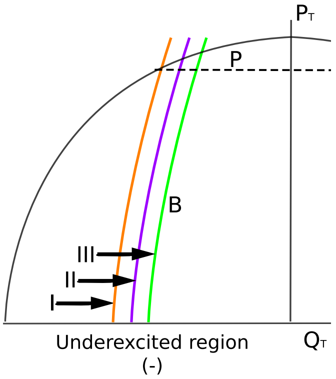


Figure 3. Coordination between UEL. I: Small-signal stability limit, II: Loss-of-excitation relay, III: Underexcitation limit set by the UEL, and P: Turbine power (IEEE 2016)

2.3 Stator Current Limiter (SCL)

A Stator current limiter is used to limit the high stator currents that cause overheating of the stator winding. High stator currents may occur due to significant changes in system voltage or increase in turbine power without considering the capability of generator stator windings. Here the SCL cannot directly limit the generator output current (stator current); it can only modify the field excitation during the operation with reactive stator current. As another option, tab change of the main transformer or reduction of turbine power can be considered to reduce the stator current.

Common SCLs mainly vary the excitation level to limit the stator current. The excitation level is varied based on whether the synchronous generator is operating inside the overexcited or underexcited region. When the generator is overexcited, the SCL should reduce the excitation in order to reduce the stator current, while when the generator is underexcited, the SCL should increase the excitation to reduce the stator current.

The SCL is responsible for limiting the stator current between points X and Y on the GCC, as shown in Figure 1, where the SCL's limit or set-point should be below the OEL's predefined limit and above the UEL's predefined limit. In addition, the SCL set-point is usually set above the stator current corresponding to generator-rated apparent power to ensure that the SCL does not reduce the excitation during the normal operation. Most commonly, the turbine capability limits the active power output of the generator in such a way, the reactive power output remains below the SCL characteristic. Thereby, the SCL would never become active under normal voltage conditions. But, if the turbine power increases, the generator stator windings should be upgraded. Otherwise the SCL might become active under normal operating conditions.

2.4 Volts-per-Hertz (V/Hz) Limiters

V/Hz limiters protect the generator's core and step-up transformers from significant overheating and damage due to excessive magnetic flux. The excessive magnetic flux typically results from low frequency and/or over-voltage. This limiter calculates the ratio of per unit voltage and per unit frequency and controls the field voltage to limit the generator voltage when the V/Hz value exceeds a preset value. The volts-per hertz limiters trip the generator by shutdown the field voltage when the V/Hz value exceeds the preset value for a certain period. V/Hz limiter usually has two grades of settings, where one with a higher V/Hz and shorter time settings, and another with a lower V/Hz and longer time settings. This is due to terminal limitations of the generators and step-up transformer.

3 Modelling of Limiters

This section gives an overview of the modelling of protective limiters based on (Kundur, Balu, and Lauby 1994) and (IEEE 2016). In addition, it provides information about implemented user interfaces in the models using Modelica. Indeed, most variables name and their descriptions in each model originate in the IEEE Std 421.5-2016 (IEEE 2016). However, some variables names and descriptions are modified for modelling purposes. Moreover, all the model's inputs are in the per-unit except the frequency which is Hz.

3.1 Field Current Overexcitation Limiter (FCOEL)

The FCOEL, modelled based on *OEL2C* in IEEE Std 421.5-2016 (IEEE 2016) shown in Figure 4, can interact with the Automatic Voltage Regulator (AVR) either as an addition to the summation point or at the takeover junction. If the FCOEL model is connected to the summation point, the maximum output limit should be set to zero, while the minimum output limit should be set to a negative value corresponding to the maximum reduction. Unlike when the FCOEL is connected to the takeover junction, the maximum output limit of the FCOEL should be set to larger values, whereas the minimum output limit should be set to a positive value that maintains the minimum excitation level. And the input to the limiter could be the generator field current I_{FD}, generator field voltage E_{FD}, or a signal proportional to exciter field current V_{FE}.

Besides, the FCOEL's output is limited by the PID controller's maximum and minimum limit if the lead-lag function is turned off; otherwise, the output is limited by $V_{\text{FCOELmax1}}$ and $V_{\text{FCOELmin1}}$ or $V_{\text{FCOELmax2}}$ and $V_{\text{FCOELmin2}}$.

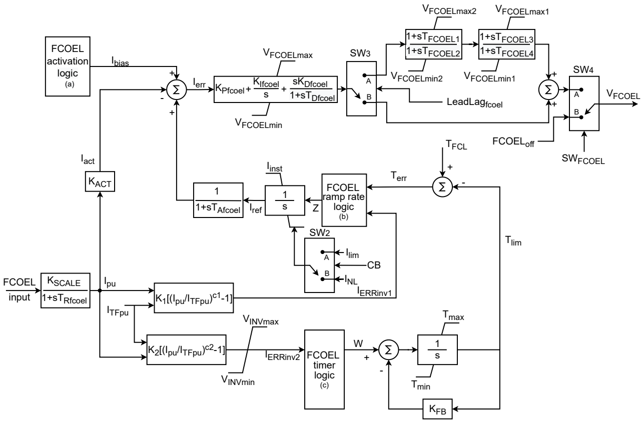


Figure 4. Block diagram of the field current overexcitation limiter (FCOEL)

The activation logic (a) in Figure 4 allows the user to specify an activation delay time T_{enFCOEL}, this time delay will disable the instantaneous FCOEL responses for a certain time to allow very high transient forcing capability. Also, it allows defining time delay to reset the limiter and reset threshold value.

When the timer error signal T_{err} = T_{FCL} - T_{lim} is less or equal than zero, or if the actual feedback field current I_{act} is greater equal than the reference I_{ref} for longer or equal than the activation delay time T_{enFCOEL}, or if the T_{enFCOEL} is equal to zero, then the output of the activation logic I_{bias} becomes zero.

Thus, the error I_{err}, the input to the PID controller is reduced, and consequently, the output of the FCOEL V_{FCOEL}

reduces towards the limiter minimum until the field current reaches the preset limit. The preset limit could be instantaneous field current limit I_{inst}, or thermal (long-term) value I_{lim} or no-load limit I_{NL} level. While, when the I_{ref} is less-equal than I_{inst} and the error $(I_{ref} - I_{act})$ is larger or equal than the reset-threshold value $I_{\text{THoffFCOEL}}$ for longer or equal than the reset time delay T_{offFCOEL}, then the output I_{bias} becomes the reset reference value $I_{\text{resetFCOEL}}$. As a consequence, the output V_{FCOEL} will reach back to maximum limits set by the PID controller or double lead-lag function.

This model comprises both instantaneous and timed responses, where the timed response could follow fixed ramp rates or inverse-time. The inverse-time characteristic of the FCOEL is calculated using the actual field current I_{pu} and parameters (K_2, c_2, and I_{TFpu}), and then the output signal I_{ERRinv2} is applied to the timer logic (c) in Figure 4. The inverse-time characteristic can be disabled by either set the parameter K_2 to zero or by setting the limits V_{INVmax} and V_{INVmin} to zero. The timer logic determines the input signal to the timer integrator by using defined fixed time ramp rates, $Fixed_{ru}$ and $Fixed_{rd}$ together with I_{ERRinv2}. Whereas the timer integrator output T_{lim} and fixed-parameter T_{FCL} determine the timed action of FCOEL.

The ramp rate logic (b) in Figure 4 uses the T_{err} signal to determine if the reference field current I_{ref} should be ramped up to the I_{inst} value or ramped down to the I_{lim} or I_{NL}. The ramp rate can be constant values as K_{ru} (ramp-up) and K_{rd} (ramp-down) or can be given by I_{ERRinv1}. Where the signal I_{ERRinv1} is calculated similarly to I_{ERRinv2} with parameters (K_1,c_1, and I_{TFpu}). Switching from instantaneous limit I_{inst} to timed limit I_lim or I_{NL} is constituted by setting the SW_1 to false and K_{ru} and K_{rd} to large values. But, if it is a desire to have a ramp down at a rate calculated from overexcitation can be obtained by setting the SW_1 to "true" and select a proper parameter for K_1 and c_1. During the no-load condition (circuit breaker is in the open position), the switch SW_2 is changed to position "B", meaning the lower limit of the integral to the no-load limit I_{NL}. Otherwise, during the normal operating condition (circuit breaker is in the closed position), the lower limit is set to I_{lim}.

There are PID and double lead-lag at the output of the FCOEL, which determine the FCOEL's dynamic response. If PID control is desired, the lead-lag functions can be disabled by changing the position of the switch SW_3 to "B". Alternatively, if the double lead-lag compensation is desire, the gain K_{Ifcoel} and K_{Dfcoel} should be set to zero, simultaneously the position of the switch SW_3 should be set to "A".

Finally, the switch SW_4 at the output of the FCOEL is used to switch off the limiter's output by the user command. Then the output of the FCOEL will be the user-defined parameter $FCOEL_{off}$. Figure 5 shows modelled FCOEL in Modelica, where the radio buttons are implemented to switch the SW_1 to alternate from fixed time ramp rate to the calculated ramp rate from overexcitation. There is a checkbox named `LeadLag_fcoel` shall be checked to change the position in SW_3 to "A" in order to activate the lead-lag function. Moreover, the lead-lag should be enabled in order to parameterise the lead-lag function else the parameters boxes are locked.

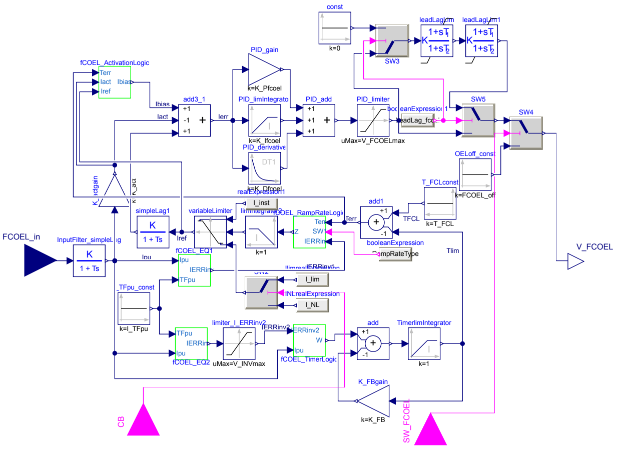


Figure 5. Implementation of Field current overexcitation limiter (FCOEL) in Modelica

3.2 Stator Current Underexcitation Limiter (SCUEL)

The block diagram of the SCUEL shown in Figure 6 is based on the type *UEL2C* in IEEE Std 421.5-2016 (IEEE 2016). The limiter senses the active power P_T and reactive current I_Q and increases the excitation when the generator runs at underexcitation below the defined characteristic value. Since the limiter is a separate circuit, the output signals of this limiter can interact either with the summing point or the High-value (HV) gate input of the excitation system. The interaction with the summing point leads to normal voltage control, whereas interaction with the HV gate (takeover junction) will overwrite the normal action of the AVR. Be aware that the inputs could also be the active current I_P instead of active power P_T, but it should be a positive value and also reactive power Q_T can be used instead of reactive current I_Q.

If the SCUEL interacts with the summation point, the minimum output limit should be set to zero, while the maximum output limit should be set to a large positive value. On the contrary, if the SCUEL is connected to the takeover junction, the maximum output limit of the SCUEL output should be set to 0, whereas the minimum output limit should be set to a significant negative value. Besides, the SCUEL's maximum and minimum limit is set by the PID controller's limit if the lead-lag function is turned off; otherwise, the output is limited by $V_{\text{SCUELmax1}}$ and $V_{\text{SCUELmin1}}$ or $V_{\text{SCUELmax2}}$ and $V_{\text{SCUELmin2}}$.

The voltage bias logic (a) in Figure 6 provides an adequate voltage ratio to be used in equation blocks F_1 and F_2. The logic can be bypassed by setting the parameter $V_{\text{biasSCUEL}} = 1$. The equation blocks F_1 and F_2 in the

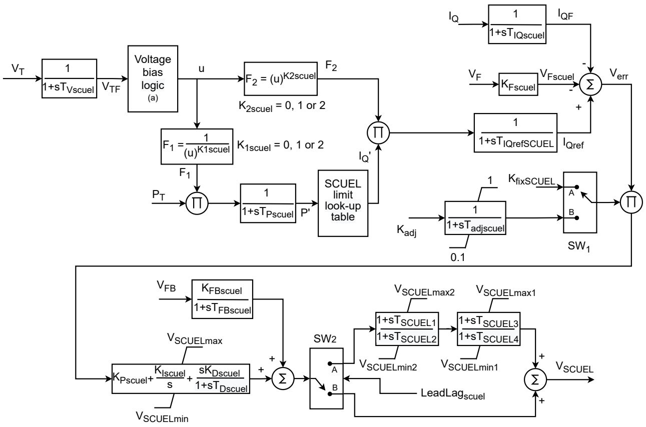


Figure 6. Block diagram of the Stator Current Underexcitation Limiter (SCUEL) (IEEE 2016)

SCUEL block diagram provide appropriate adjustments so that the effects of the terminal voltage V_T on the limiter are taken into account. The adjustments provided by the F_1 and F_2 are based on the limiting characteristics of the SCUEL and determined by the constants $K_{1\text{scuel}}$ and $K_{2\text{scuel}}$. If the SCUEL is configured to be influenced by the active and reactive currents, the limiter characteristic is set proportional to V_T by using the $K_{1\text{scuel}} = K_{2\text{scuel}} = 1$. While, if the limiter is influenced by the active and reactive components of the apparent impedance looking from the machine terminals, the characteristic can be set to proportional to the V_T^2 by using $K_{1\text{scuel}} = K_{2\text{scuel}} = 2$. However, the latter limiting characteristic requires proper coordination with generator protection functions such as loss-of-excitation relays. Also, this function can be disabled by using $K_{1\text{scuel}} = K_{2\text{scuel}} = 0$.

The SCUEL shown in Figure 6 takes the active power P_T and multiplies it by F_1, further filters it and the resulting normalised value P' is sent to the look-up table. The limiting characteristic defined in a lookup table determines the corresponding normalised reactive current value I'_Q related to P'. The reference I_{Qref} is determined by multiplying the I'_Q and F_2 and then compared with the filtered actual reactive current I_{QF}. If the error signal $V_{err} = I_{\text{Qref}} - I_{\text{QF}} - V_{\text{Fscuel}}$ becomes negative under the normal condition, the limiter's output will be the minimum PID or lead-lag limit, meaning no actions are taken. When the error signal becomes positive, the output of the SCUEL drives in the positive direction and boosts the excitation to move the operating point back towards the SCUEL limit.

The SCUEL reduction gain can be either automatically adjusted (depending on V_T, P_T and I_Q) or can have a fixed constant gain value. To be able to enable the automatically adjusted gain, the logic switch SW_1 should be selected to position "B", where the automatic adjustable gain reduction K_{adj} is calculated using Equation 1, while the fixed constant gain, given by the parameter K_{fixSCUEL} value, is enabled by switching the SW_1 to position "A". The gain reduction can be disabled by setting $K_{\text{fixSCUEL}} = 1$.

$$K_{adj} = \frac{\dfrac{V_T^2}{X_q} + I_Q}{\sqrt{(\dfrac{V_T^2}{X_q + I_Q})^2 + P_T^2}} \tag{1}$$

where

K_{adj}:	Automatic adjustment gain	[-]
V_T:	Generator terminal voltage	[pu]
P_T:	Active power	[pu]
I_Q:	Reactive current	[pu]
X_q:	q-axis synchronous reactance	[pu]

The excitation stabiliser signal from the AVR shall be provided to the input V_F, and it helps to damp the oscillations. The input V_{FB} can only be used in conjunction with the excitation system ST7C model. As mentioned earlier, the lead-lag blocks can be disabled by changing the position of the switch SW_2 to "B", and PID control by set gain K_{Iscuel} and K_{Dscuel} to 0. Besides, if the lead-lag function is desired, change the SW_2 to position "A", and the time constants should be appropriately adjusted to provide sufficient damping. The limiting characteristic of the SCUEL is depicted in Figure 7, where the limit is composed of four straight line segments. All five endpoints should be defined in terms of P_i and I_{Qi} values in order to determine the limiter characteristic. For any values of P', the corresponding value of I'_Q between the segment endpoints is determined using linear interpolation.

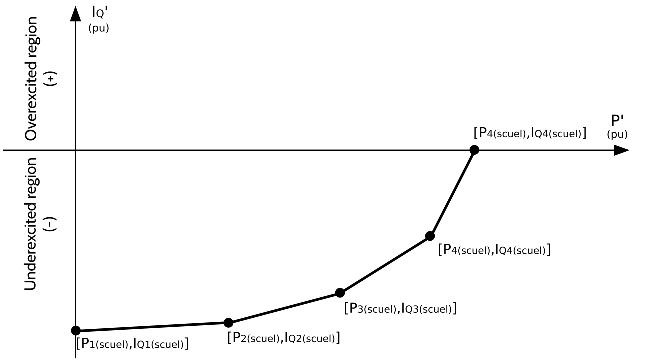


Figure 7. Normalised limiting characteristic for the SCUEL (IEEE 2016)

Figure 8 displays the modelled SCUEL in Modelica. There are two checkboxes implemented, one that changes the position of SW_2 from "B" to "A", while the other one alternates the switch from position "A" to "B" in order to enable the adjustable gain reduction, by checking it. Additionally, by enabling the lead-lag function and adjustable gain reduction, the parameter boxes will be enabled to allow the user to define corresponding parameters; otherwise, the parameter boxes are locked. Besides, when the automatically adjustable gain reduction is enabled, the parameter box for K_{fixSCUEL} will be locked. Alternatively, by unchecking the checkbox, the parameter box for the fixed

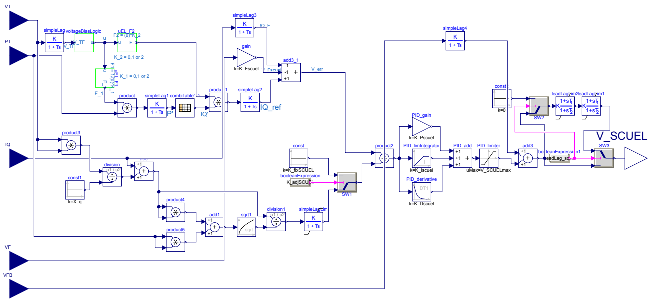


Figure 8. Implementation of Stator current underexcitation limiter (SCUEL) in Modelica

gain K_{fixSCUEL} will be opened, and the switch is back to the position "A".

3.3 Stator Current limiter (SCL)

A stator current limiter modifies the excitation level to reduce the reactive component of the stator current; as a consequence, the stator current will be limited. Figure 9 shows a block diagram of the SCL based on the type *SCL1C* in IEEE Std 421.5-2016 (IEEE 2016). This limiter uses the stator current I_T, reactive current I_Q, and reactive power Q_T at the generator terminal as inputs. Further, the output signal V_{SCL} from the SCL can only interact with the summing point of the excitation system.

When the magnitude of the I_T becomes greater than the adjustable pick-up value I_{SCLlim}, then the SCL starts to influence the excitation after the time delay. The time delay before limiting allows a short-term increase of stator current during a system disturbance or startup. There are three types of time delay functions implemented in this limiter. One of the time delay functions caused by the transducer delay in the measurement of the stator current is represented by the time constant T_{IT}. The second and third type time delay functions are enabled if the switch SW_1 on position "B", and the time delays are determined by an inverse time characteristic T_{INV} or a fixed-time T_{DSCL}. The switch SW_2 in the delayed reactive power logic (c) in Figure 9 should be set to "true" to enable the inverse time delay; else, the fixed-time delay will be applied.

A deadband is implemented at unity PF because the SCL does not affect the active power, so modifying the excitation level does not give any benefits at unity PF. There are two options to provide the deadband to the limiter; if the reactive current is used to modify the excitation, the deadband zone can be defined by the parameter I_{Qmin}. Whereas if the reactive power is used, then the deadband can be defined by the parameter V_{SCLdb}.

When the SW_1 is in position "A", the reactive current is used to determine if the generator is operating in an overexcited or underexcited condition. When the SW_1 is in position "B", the reactive power is used to determine the generator's operating condition. During the underexcited condition, the excitation current is increased, unlike during the overexcited condition, the excitation current is decreased.

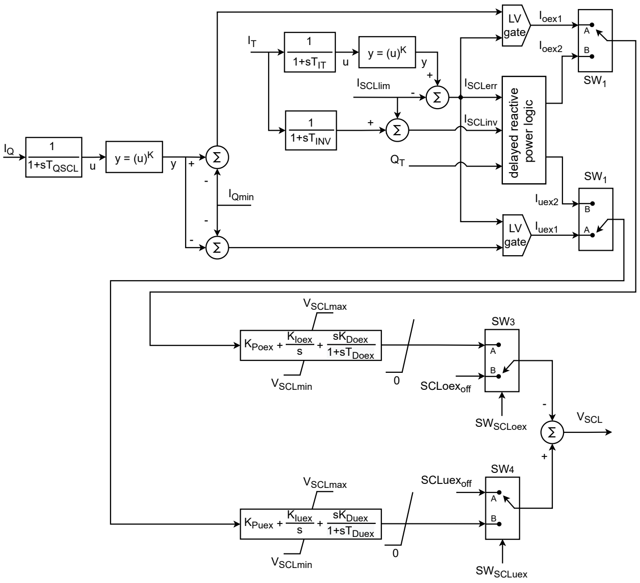


Figure 9. Block diagram of the stator current limiter (SCL) (IEEE 2016)

There are two individual PID controllers for overexcited and underexcited control loops, which can be individually adjusted for proper tuning of the PID controllers. If I_Q or Q_T within the deadband, the input to the PID controller becomes zero, and the PID controller's output will be held constant. Also, during the normal operation condition (when the I_T is lower than the I_{SCLlim}), the outputs of the overexcited and underexcited range will be zero; consequently, the output of the output V_{SCL} will be zero. When the I_T is higher than the I_{SCLlim}, the output V_{SCL} reduces during the overexcited range, while V_{SCL} increases during the underexcited range. However, under both ranges, the output decreases or increases until the reactive current or power reaches the deadband zone or until the I_T or Q_T becomes equal to the I_{SCLlim}. And there are two switches SW_3 and SW_4 at the output of the overexcited and underexcited range used to switch off each range's output, respectively, where the user shall give the command. When the outputs of the overexcited and underexcited range are switched off, the user-defined parameters `SCLoex_off` and `SCLuex_off` will be the output of each region, respectively.

The modelled stator current limiter in Modelica is displayed in Figure 10, where the switches at the output SW_3 and SW_4 should have a Boolean signal to enable the limiter's output.

Furthermore, there are implemented radio buttons for the switches, SW_1 and SW_2. If the radio button called "Reactive current controller" is selected, then the SW_1 is changed to position "A". Whereas, if the "Reactive power controller" radio button is selected, then the SW_1 is shifted to position "B", and the SCL uses the reactive power to determine the operating condition of the generator. As well, the switch SW_2 can be set to "true" by selecting the radio

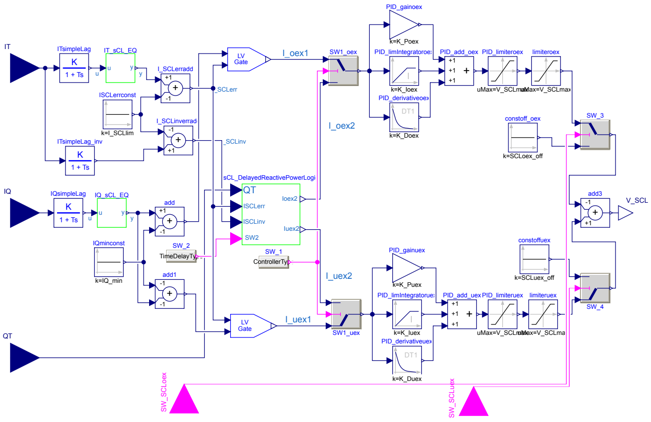


Figure 10. Implementation of stator current limiter (SCL) in Modelica

button "Enable inverse time delay", while it can be set to "false" by selecting the "Enable fixed-time delay".

3.4 Volts-per-Hertz (V/Hz) Limiter

V/Hz limiter is a voltage limiter that limits the voltage function of frequency. This model is build based on (Kundur, Balu, and Lauby 1994) and interacts with the AVR at the summing point, and reduces the reference so that the terminal voltage reduces with respect to the frequency reduction. A block diagram of the V/Hz limiter is shown in Figure 11, where the limiter takes inputs as terminal voltage V_T and frequency f. The limiter calculates the ratio between the terminal voltage and the frequency in per unit, which then is compared with the limiting value V_{ZLM} to determine the error E_{rr}.

If the E_{rr} is greater than zero, a timer will start to count, and when the time is greater than T_d, the E_{rr} signal will be sent further to the PID controller. During the normal operation (when the limiter us inactive), the E_{rr} is negative and thus, the output is zero; while when limiter is active and the E_{rr} becomes greater than the zero, the output starts to increase until the voltage reduces and the E_{rr} becomes less-equal than the zero. Additionally, this limiter can be activated and deactivated by the Boolean signal, and when the limiter is deactivated, the output will be zero.

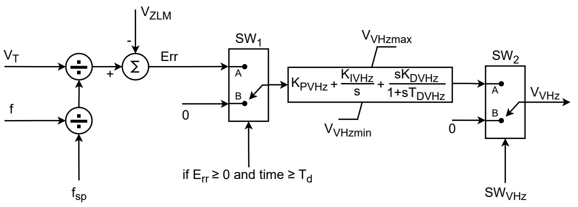

V_T
f
f_sp
V_ZLM
-
+
Σ
Err
SW_1
A
B
0
K_PVHz + K_IVHz/s + sK_DVHz/(1+sT_DVHz)
V_VHzmax
V_VHzmin
SW_2
A
B
0
V_VHz
if E_rr ≥ 0 and time ≥ T_d
SW_VHz

Figure 11. Block diagram of the volts-per-hertz (V/Hz) limiter (Kundur, Balu, and Lauby 1994)

A V/Hz limiter modelled in Modelica is shown in Figure 12, where the dashed lines illustrate the conditional connections that the user can activate either the constant frequency set-point f_{sp} or variable frequency set-point f_{vsp}.

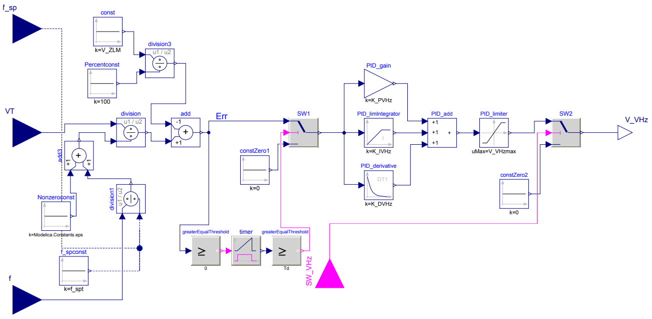


Figure 12. Implementation of V/Hz limiter in Modelica

There is a checkbox called `Enable_fvsp` implemented. By checking the checkbox the real input f_{vsp} is enabled so the variable frequency input can be connected to the limiter, and simultaneously the f_{sp} will be disabled. When the checkbox is unchecked, the f_{vsp} real input dissipates, and the f_{sp} is enabled. Additionally, the switch SW_2 at the output allows the user to activate and deactivate the limiter by a Boolean signal. Besides, a constant epsilon is added with the frequency to protect against division by zero problems.

4 Simulations Results

This section presents simulation results of Field Current Overexcitation Limiter (FCOEL), Stator Current Underexcitation Limiter (SCUEL), Stator Current Limiter (SCL), and the Volts-per-Hertz limiter (V/Hz).

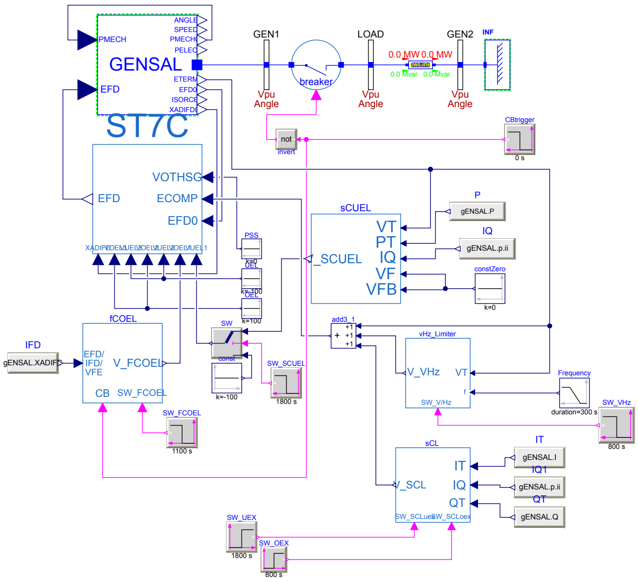


Figure 13. Test setup

The test setup is created using a GENSAL generator, transmission line, infinite grid, and excitation system type*ST7C* from the OpenIPSL version 2.0.0 (ALSETLab et al. 2018), as shown in Figure 13. The system power base and frequency for all the components are set to $10\,MVA$ and $50\,Hz$, accordingly. The generator is initialised, as presented in Table 1, during the various simu-

Table 1. Initialisation of GENSAL generator for simulation

Parameters	*Description*	*Values*	*Units*
P_0	Initial active power	-	MW
Q_0	Initial reactive power	-	Mvar
v_0	Initial voltage magnitude	1	pu
$angle_0$	Initial active power	0	°
ω	Initial active power	0	pu

lations. Since the initial active power P_0 and the reactive power Q_0 will be varied for different simulation tests, the values are not presented in the table.

4.1 Field Current Overexcitation Limiter (FCOEL)

The test is performed by enabling the limiter output at 1100 s and open the circuit breaker at 2500 s.The initial active power P_0 and reactive power Q_0 of the generator are set to $8\,MW$ and $6\,Mvar$, accordingly.

The test results of the field current overexcitation limiter are presented in Figure 14. After the startup, the generator field current I_{FD} is stabilised to $2.15\,pu$, but the output of the FCOEL V_{FCOEL} is continuously at 100. When the limiter output is activated, the field current is limited to $I_{lim} = 1.95\,pu$, and consequently, the V_{FCOEL} reduces. Further, when the circuit breaker opens, the field current is limited to $I_{NL} = 1.07\,pu$, as desired.

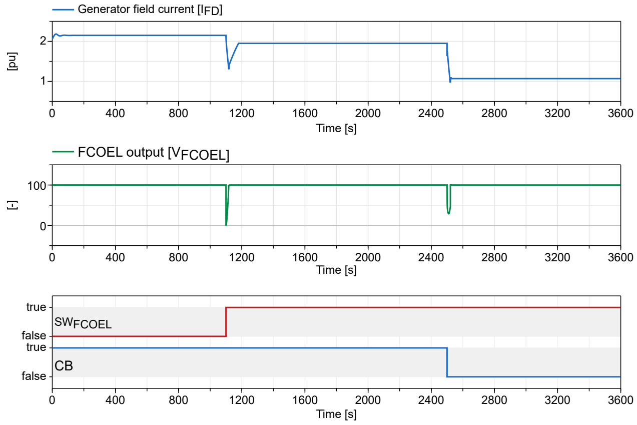


Figure 14. Field current overexcitation limiter (FCOEL) performance

4.2 Stator Current Underexcitation Limiter (SCUEL)

There is a switch manually added to the output of the SCUEL to control the output of the limiter (see Figure 8). The switch's position is set to change at $1800\,s$ to enable the output of the SCUEL. The initial active power P_0 and reactive power Q_0 of the generator are set to $7.5\,MW$ and $-4\,Mvar$, respectively, during the SCUEL simulation test. Besides, the inputs V_F and V_{FB}, are set to zero due to a lack of outputs from the *ST7C* model.

Initially, the limiter output is not connected to the AVR and the reactive current I_Q in a steady-state around $-0.36\,pu$, while the underexcitation I_Q limit is set to $-0.25\,pu$ at $0.75\,pu$ active power P_T. As a result, the limiter tries to increase the field current by increasing the output of the limiter towards the maximum limit (see Figure 15). When the limiter output is connected to the AVR at $1800\,s$, the limiter gradually decreases the output; hence the reactive current increases, and it ends up at $-0.25\,pu$, as expected. The output of the limiter is decreased and stabilised to a signal value of -99.98.

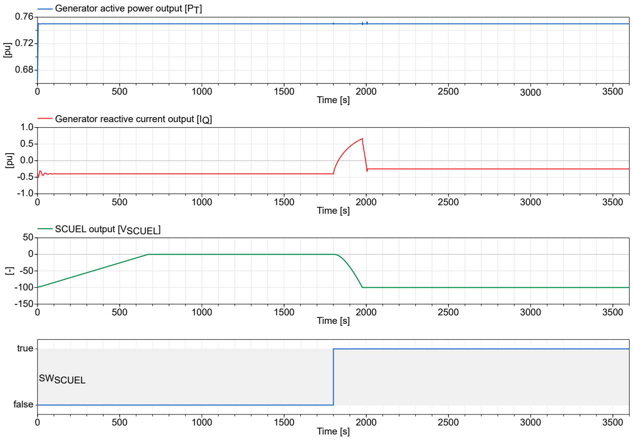


Figure 15. Performance of stator current underexcitation limiter (SCUEL)

4.3 Stator Current limiter (SCL)

The simulation test for the SCL is performed in two operational regions, inside the overexcited and the underexcited region. Both regions have separate PID controllers to operate under each region. As mentioned in subsection 2.3, SCL also consists of two types of controllers, reactive current and reactive power controllers. And each controller is also examined under each operational region. The output of the overexcited and underexcited regions is controlled by the Boolean input signals SW_{OEX} and SW_{UEX}, respectively, and these are set to "true" at 800 and $1800\,s$. In order to obtain generator terminal current IT above the pickup level I_{SCLlim} during the overexcited region, the initial active power P_0 and reactive power Q_0 of the generator are set to $8\,MW$ and $8\,Mvar$, accordingly. Whereas during the underexcited region test, only the initial reactive power Q_0 of the generator is changed to $-8\,Mvar$. In addition, the reactance of the transmission line X is set to zero during the simulations.

Figure 16 shows the performance of SCL during the overexcited region, where the solid lines illustrate reactive power controller performance while the dashed lines illustrate reactive current controller performance. Both controllers are simulated with similar parameters, except that each controller's proportional and integral gain are

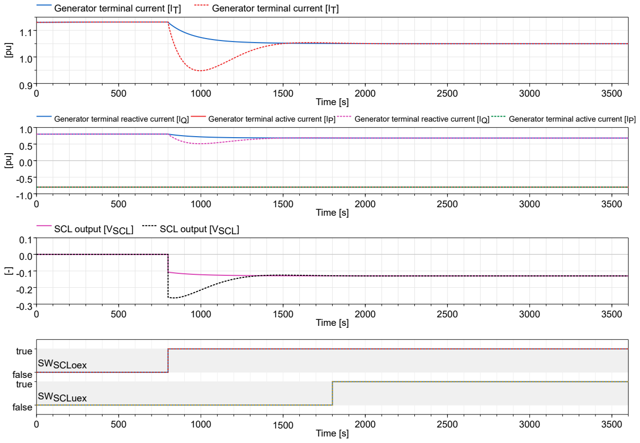


Figure 16. Test setup for stator current limiter (SCL) model

tuned separately to secure proper limitations. Note that the PID controller should be tuned separately for the reactive power and current controller. The overexcited region's output is enabled at 800 s, whereas the underexcitation region's output is enabled at $1800\,s$. The stator current limiter output V_{SCL} is reduced immediately after the overexcitation output is enabled and causes the field current to be reduced. As a consequence, the reactive power or current output of the generator reduces, thereby the generator terminal current I_T to reduced from $1.11\,pu$ towards the threshold value of $1.05\,pu$, as anticipated. However, enabling the output of the underexcitation region does not cause any reasonable changes. Furthermore, there is a notable difference between the performance of the reactive power and the current controller. As the results show that the reactive power controller reduces the terminal current smoothly; thereby, the overshoot that occurs during the reactive current control is eliminated.

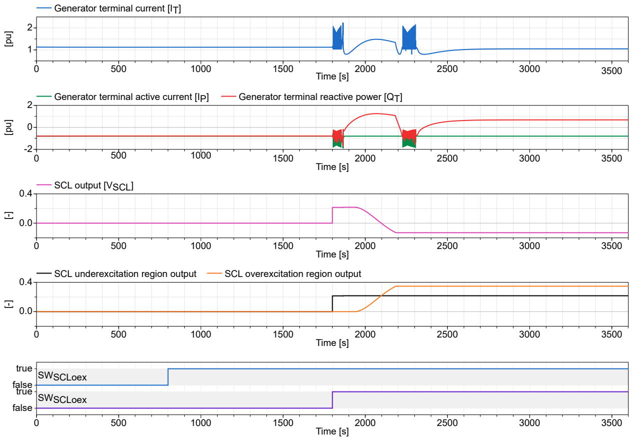


Figure 17. Test setup for stator current limiter (SCL) model

The simulation results of the reactive power controller in SCL during the underexcited condition are illustrated in Figure 17. The test is performed similarly to the overexcitation simulations when the Boolean signal SW_{UEX} turns "true" at $1800\,s$, the output V_{SCL} is increased, and causes the reactive power to increase into the overexcitation region. Hence, the SCL overexcitation part takes control of the reactive power and reduces until the terminal current reaches within I_{SCLlim}, as desire. There are some oscillations when the reactive power starts to increase into the overexcitation region as well as when the reactive power is reduced into the underexcitation region. The simulation results of the reactive current controller of SCL during the underexcited condition are not presented due to the random oscillations.

4.4 Volts-per-Hertz (V/Hz) Limiter

There is a ramp logic that is connected to the V/Hz model that represents actual frequency and a Boolean signal input that controls the output of the limiter (see Figure 13). At $800\,s$, the Boolean signal turns "true", so the output of the V/Hz limiter gets enabled, and then at 1800 s, the ramp logic reduces the frequency from nominal frequency $50\,Hz$ to $45\,Hz$. The generator is initialised with parameters $P_0 = 8\ MW$ and $Q_0 = 5\,Mvar$.

Figure 18 illustrates the performance of the V/Hz limiter, where the terminal voltage in the steady-state at $1.086\,pu$ before the set-point changes. When the frequency reduces, the deviation between voltage and frequency will increase above the defined limit; thus, the limiter's output signal starts to increase. As a consequence, the voltage starts to decrease nicely and stabilised at $0.99\,pu$ as expected.

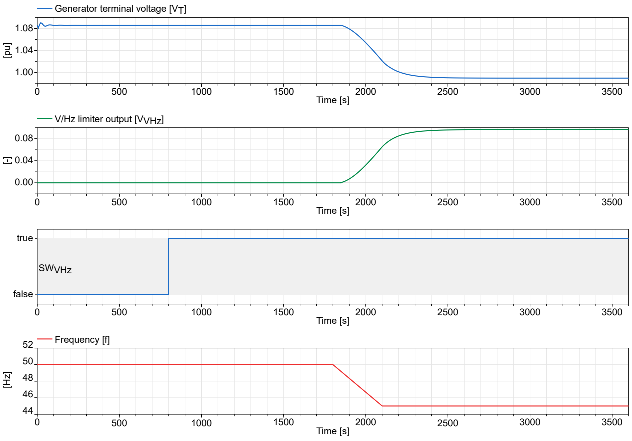


Figure 18. Performance of volts-per-hertz limiter

5 Discussion

This paper aims to model protective limiters Field current Overexcitation limiter (FCOEL), Stator current underexcitation limiter (SCUEL), Stator current limiter (SCL), and Volts-per-hertz (V/Hz) limiter in the Modelica modelling language. Fundamentally, the limiters are modelled based on IEEE Std 421.5-2016 (IEEE 2016); however, the models have been modified a bit due to modelling purposes and complexities. While the excitation system, type ST7C, is obtained from the OpenIPSL library, and the V/hz limiter is modelled based on Kundur (Kundur, Balu, and Lauby 1994).

Since the primary focus of this paper is to model the protective limiters, the test setup modelling is kept simple as possible to analyse the modelled excitation control system's performance. However, it should be properly modelled in the future for better examination of the model's performance.

According to IEEE Std 421.5-2016 (IEEE 2016), the lead-lag function in FCOEL and SCUEL could be disabled by setting their time constants to zero; however, it causes errors during the simulations; thus, those time constants are set to constant epsilon to be able to simulate. Despite this, the simulations failed randomly; this problem is solved by adding the switches to bypass the control signal from the PID controller to the output.

A PID controller is implemented in SCUEL and SCL, even if it is not stated in IEEE Std 421.5-2016 (IEEE 2016), in order to provide the user access to tune the controller's output properly.

According to IEEE Std 421.5-2016 (IEEE 2016), one of the FCOEL inputs could be generator field voltage E_{FD} as described in subsection 2.1, but during the simulations, using the E_{FD} as an input cause simulation error. The reason could be that in the FCOEL model, the signal I_{act} obtained from the input and connected to the output, causing algebraic loop. But the simulation error may be eliminated by adding a delay in the signal I_{act} or at the input.

The V/Hz limiter model is fundamentally modelled based on Kundur (Kundur, Balu, and Lauby 1994) as described in Section 2.4; however, simulation results showed that the model's behaviour was not desirable. Hence, the lead-lag function of the limiter was replaced by the PID controller, and the gains were removed to get a reasonable behaviour.

The overall behaviour of all the models was reasonable; be aware that the proportional and the integral gain of the FCOEL is relatively high, and the reason is not apparent yet; however, the model works as desire.

There are some oscillations when the reactive power starts to increase into the overexcitation region as well as when the reactive power is reduced into the underexcitation region (see Figure 17); the reason could be that the test grid is not adequately modelled in order to tackle the high current underexcitation operation. The simulation results of the reactive current controller of SCL during the underexcited condition are not presented in subsection 2.3 due to the random oscillations for the same reason as mentioned before. The SCL model should be further analysed in the future with a proper grid model to avoid unwanted oscillations. Further, by reflecting on the SCL's simulation results, the power controller may be the desired controller since it has the advantage of time delay and deadband, also work smoothly. However, better and smooth control of the reactive current controller could be achieved by better tuning of the PID controller.

The FCOEL, SCL, V/Hz limiters models have a switch at the output to disable the control function. These switches are used to enable the output during the simulation, which results in a sudden change in the control signal. This sudden increase in the control signal also impacted for example, field current, active power, or reactive power output. The main reason might be explained that the PID controller pushed the maximum control signal to the output, and the output signal does not make any changes in the system; therefore, when the switches enable the outputs, the maximum control signal is applied instantaneously. One possible way to fix this problem is by adding a self-reset function for the PID controller to reset when the output disables.

6 Conclusions

In this paper, the excitation control system's limiters are mainly object-oriented modelled in Modelica modelling language using Dymola software. The models are fundamentally modelled with reference to IEEE Std 421.5-2016 (IEEE 2016) and Kundur (Kundur, Balu, and Lauby 1994).

The limiters Field current overexcitation limiter (FCOEL), Stator current underexcitation limiter (SCUEL), Stator current limiter (SCL), and Volts-per-hertz (V/Hz) limiter were modelled separately from scratch, except the AVR, obtained from the external library OpenIPSL. Later, the models mentioned above were simulated separately and then compared to these controller's and limiter's theoretical behaviour.

In conclusion, all the models performed as desired but still need proper tuning and further development to enhance the performance. For the future it is planned to run further tests with real power plant data in order to improve and verify the behaviour of the limiter models.

References

ALSETLab et al. (2018-01). "OpenIPSL: Open-Instance Power System Library - Update 1.5 to "iTesla Power Systems Library (iPSL): A Modelica Library for Phasor Time-Domain Simulations"". In: *SoftwareX* 7, pp. 34–36. ISSN: 23527110. DOI: 10.1016/j.softx.2018.01.002.

IEEE (2016). *IEEE Recommended Practice for Excitation System Models for Power System Stability Studies.* New York: IEEE. ISBN: 978-1-5044-0855-4. URL: http://ieeexplore.ieee.org/servlet/opac?punumber=7553419 (visited on 2021-04-26).

Kundur, P., Neal J. Balu, and Mark G. Lauby (1994). *Power System Stability and Control.* The EPRI Power System Engineering Series. New York: McGraw-Hill. 1176 pp. ISBN: 978-0-07-035958-1.

Modelica Association (2017-04-10). *Modelica – a Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.4.* Linköping: Modelica Association. URL: https://www.modelica.org/documents/ModelicaSpec34.pdf.

TMCC (2019). *OpenHPL.* Version 1.0.0. University of South-Eastern Norway. URL: https://openhpl.simulati.no.

An Approach for Reducing Gas Turbines Usage by Wind Power and Energy Storage

Nejm Saadallah[1] Yngve Heggelund[2]

[1]Norwegian Research Centre, {nsaa,ynhe}@norceresearch.no

Abstract

Offshore oil and gas platforms can accelerate their shift towards lower greenhouse gases emissions by combining wind power generation and energy storage systems. However, the variability of the wind resources and power demand, and the limited storage capacity make the design of the system particularly challenging. We present a modelica library at its early stage of the development and yet with promising results. The model will be used to simulate the dynamics of the whole system for a long period of time using simplified power components and balanced micro grid. This paper shares some preliminary results by applying the model on a dataset of wind resources and power demand from the North Sea. The results are the power system dynamics and an approximation of the mass of greenhouse gas emissions. On the long term the model will be applied to determine the right control and optimization strategy to control the energy system towards lower greenhouse gas emissions without compromising the balance between the power supply and demand.

Keywords: power systems, modelica, oil platform electrification

1 Introduction

Offshore wind resources in Norway were estimated to be close to 12000 terawatt hours per year as per 2018 (Bosch, Staffell, and Hawkes 2018). This great energy potential combined with the increasing offshore wind turbine's power generation offer good opportunities to reduce the amount of greenhouse gases. Emissions are mainly due to gas turbines power generation on offshore oil and gas platforms. Most of today's oil and gas platforms are powered using up to three gas turbines. This has shown to be a reliable solution but comes at the cost of important amounts of CO2 and NOx emissions (SSB 2020). In addition, introducing variable energy sources like wind power without energy buffers, combined with the fact that gas turbines take a significant time to get started means that they have to be kept at their idle state (ready to generate power) for long periods of time. Unfortunately, this leads to release of greenhouse gas emissions without even providing power. In this paper we explore the possibility to offload gas turbines by integrating wind turbines and energy storage systems. This paper builds on earlier work on integrating offshore wind farms with nearby oil platforms (He et al. 2010), by adding an energy storage capacity, and dynamical model based on energy balance. The objective is to provide a simple to configure and fast to run modelica model (ELOGOW 2021) that can be used to quantify the reduction of greenhouse gas emissions under realistic operational conditions over a long period. The case is given by a dataset for the power demand, and wind resources (ERA5 2020). The variability of the wind resources and the demand makes it challenging to keep the system in balance. Part of our approach is to exploit possible imbalance (deficit, or losses) to qualify a design composed of physical and control sub-systems. On the long terms, the model can be used to provide decision support based on the relation between wind resources, power generated, energy storage capacity, and control strategy to reduce greenhouse gas emissions. This model is intended to be generic in the sense where it does not require detailed component design.

2 Model

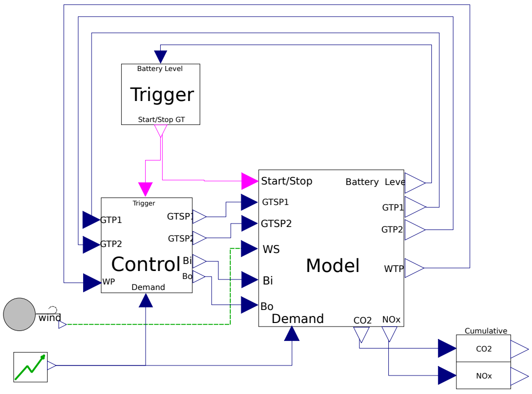


Figure 1. Simulation Overview

2.1 Simualtion Overview

Figure 1 shows the three main blocks used in the simulation. The "Model" block simulates the dynamic of the micro grid and all the components that are connected to it. This block will be further explained in the following sections. The control system strategy relies on the battery level provided by the "Model" block, and the setpoints to control the power supply by the gas turbines and the battery as well as the power for charging and draining the battery. The "Trigger" block defines two limits: a low limit and a high limit. When the low limit is reached (30 % battery level in this paper), a signal is emitted to warmup the gas turbines, and the gas turbines' power setpoints are chosen in a way to balance the power demand, plus a supplementary power to recharge the battery. On the other hand, when the high limit is reached (60 % in this paper), the gas turbines are turned off, and the battery takes over the required power supply. Note that we use this approach for testing the model, and possible improvements will be discussed later in this paper.

2.2 Physical Components Overview

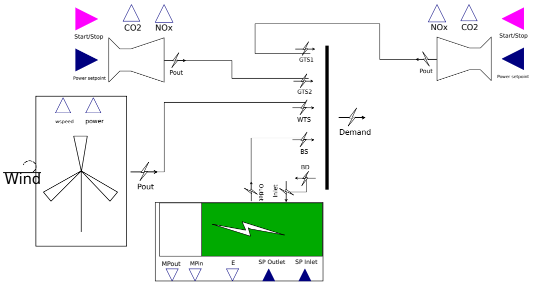


Figure 2. Physical Components Overview

The "Model" block from Figure 1 consists of the power components shown in Figure 2. The model includes two gas turbines, a wind power system configurable with zero or several wind turbines (no wake simulation yet), an energy storage system analogous to a battery, and a micro grid for energy balance. The model requires a wind source, a power demand, and set points to control the gas turbines and the energy storage system. These components are explained in the following sections.

2.3 Wind Source and Power

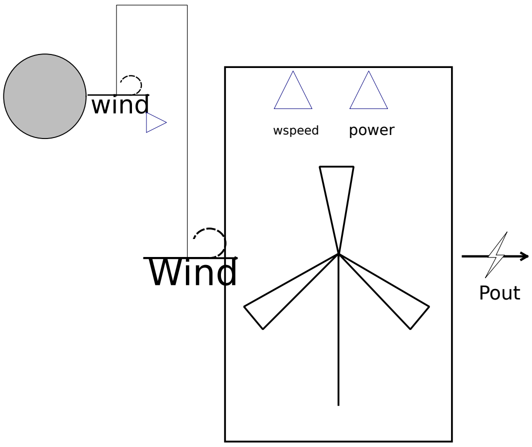


Figure 3. Wind Turbine Model

The wind turbine model in Figure 3 is purely data driven. It relates wind speed to generated power. This submodel also includes a configurable sensor model with the possibility to introduce errors in both the measured wind speed and the measured power. In this paper we rely on the 10 MW DTU wind turbine (Bak et al. 2013), and a wind speed dataset from ERA5 (ERA5 2020). The dynamic of the wind turbine system is shown in Figure 4 for a period of 7 days.

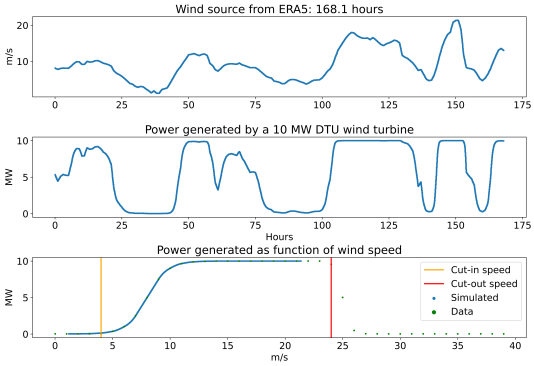


Figure 4. Wind Turbine Dynamic

2.4 Storage System

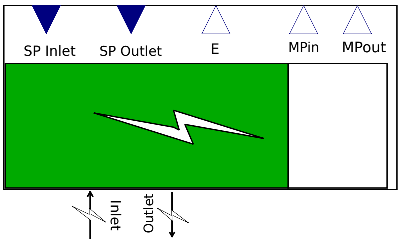


Figure 5. Energy Storage Model (Battery)

The storage model acts as a chargeable battery as shown in Figure 5. The model relates the charging power "SP Inlet" and the discharge power "SP Outlet" to the energy stored in the battery given as a % of a maximum energy. The outputs are in the physical domain (power) which will be connected to a power grid. (see Section Grid). An illustration of the dynamic of the storage system is shown in Figure 6. The initial charge is set to 30 %. In this illustration, the battery starts supplying power when its level reaches 60 % (signal to turn off gas turbines) and starts charging when its level is below 30 % and the gas turbines have started (signal to turn on the gas turbines).

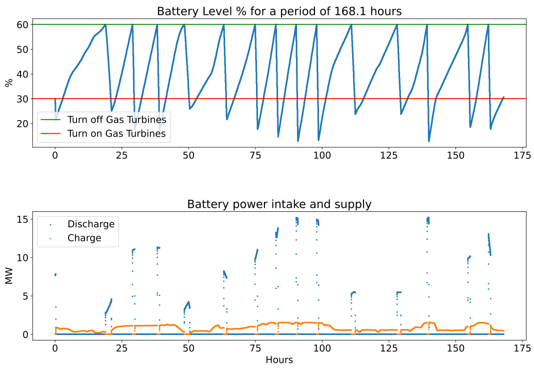


Figure 6. Illustration of the battery dynamic

2.5 Gas Turbine

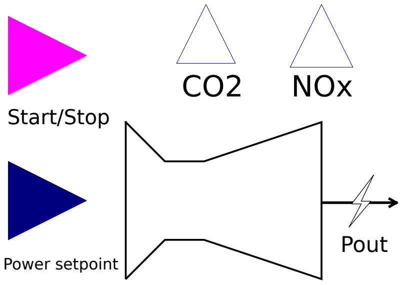


Figure 7. Gas Turbine Model

The model shown in Figure 7 simulates the dynamic of a gas turbine in a very simple form compared to more detailed models such as (Aguilera et al. 2019) and (Pires et al. 2018) . The model receives two signals a start/stop signal and a power set point. The generated power in the physical domain is driven by a configurable startup duration and a ramp up controller. The model also outputs CO2 and NOx emission rates (kg/s) according to function relating power to emission curves shown in Figure 8. The plot shows the rates at which emission gases are released for two gas turbines operating between 5 and 8 MW being the operational range used in this paper.

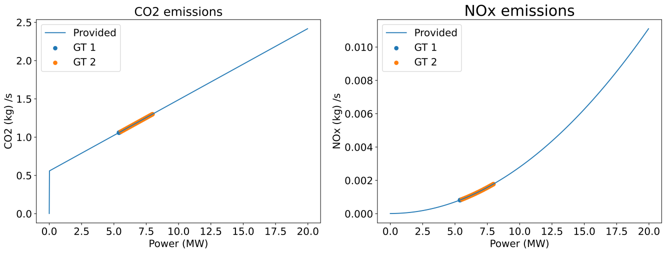


Figure 8. Gas Turbine Emission Curves

2.6 Micro Grid Balance

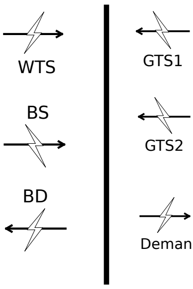


Figure 9. Micro Grid Model

The micro grid model in Figure 9 aims to keep the demand equal to the supply at any time during the simulation. This is achieved by allocating all the power surplus or deficit to a so-called “Losses” variable. An illustration of the grid dynamic is shown in Figure 10, with some minor power deficit at some time intervals (near 83 hours for example). Note that power deficits and losses are also indicators of the quality of the control system, which in this case suggests improvement. Nevertheless, the model remains consistent as the energy is kept balanced at the grid level. From a simulation perspective the power losses and deficits can be used as indicators for the microgrid stability. These could be further exploited to extent the simulation capabilities towards the electrical domain by relating power disbalance to frequency drop and gain as addressed in (Zografos, Ghandhari, and Eriksson 2018).

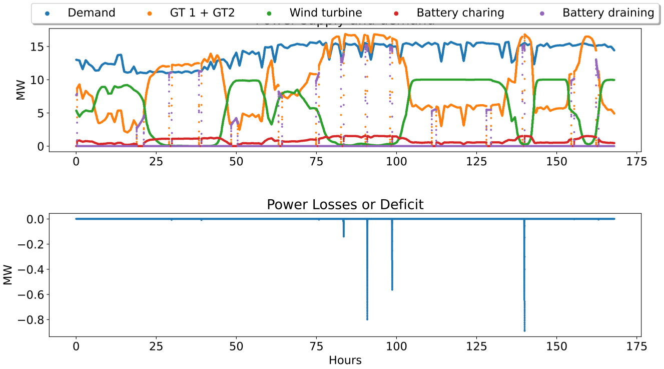


Figure 10. Illustration of the grid dynamic

3 Simulation Cases

We vary the number the of wind turbines between 0 and 4 and plot the power supplied or consumed by each component, the oil platform power demand, and the amount of gas emissions. The parameters used to run the simulations are summarized in Table 1.

3.1 Parameters

Table 1. Sizes of compiler phases, lines of code.

Parameters	*Values*
Trigger low limit	30 %
Trigger high limit	60 %
Battery max capacity	22 MW
Battery charging rate	1
Battery discharge rate	1
Battery Initial charge	30 %
Wind Turbines count	[1...4]
Wind Turbine power	File
Gas Turbine max	13 MW
Gas Turbine startup time	15 min
Time step	1s
Number of steps	604,800s (7 days)

Note. Assuming a floating structure of type semisubmersible such as the one defined in (Robertson et al. 2014) and a lithium ion battery system with an energy density of 163 Wh/kg. By allocating 1 % of the semisubmersible weight for the battery system, we roughly end up with a maximum energy of 22 MWh.

3.2 Results and Interpretation

The simulation without wind turbine in Figure 11 defines the base case, showing the situation as it is today. The whole power generation is provided by two gas turbines that are run synchronously. The simulation shows that 7 days of operations generate up to 1459 tonnes of CO2 mass, and 1.66 tonnes of NOx. In the scenario with only 1 wind turbine from Figure 12, we can notice that the battery charges and discharges frequently, because 1 wind turbine does not deliver sufficient power for the given demand, ending with 1085.99 tonnes of CO2 and 0.54 tonnes of NOx. With 2 wind turbines (Figure 13) on the other hand, we can already see that the battery stays at full charge for a relatively long periods, and even longer with 3 and 4 wind turbines (Figure 14 and Figure 15), with even fewer CO2 and NOx releases. However, the energy losses become important in the scenario with 4 wind turbines suggesting that the design of the system is oversized. Figure 16 and Figure 17 summarize the energy contribution from each component and the emitted masses of CO2 and NOx.

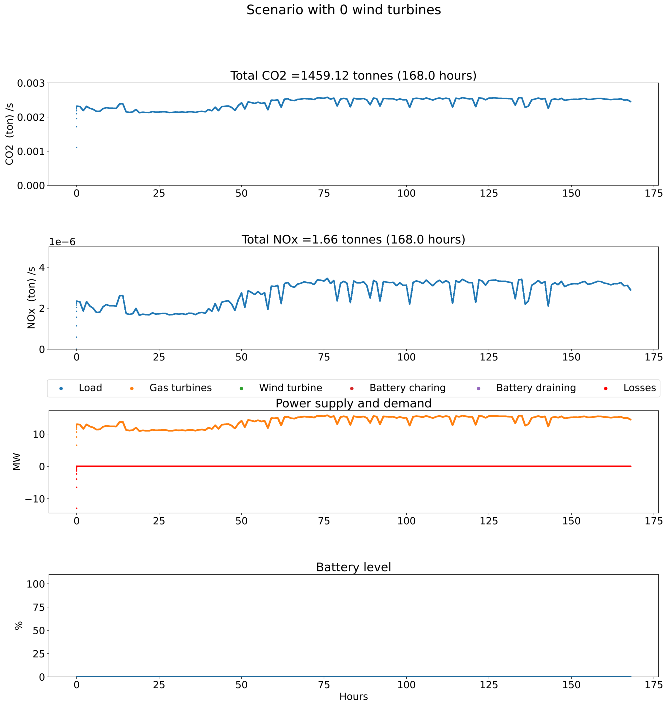


Figure 11. Base case without wind turbines and battery

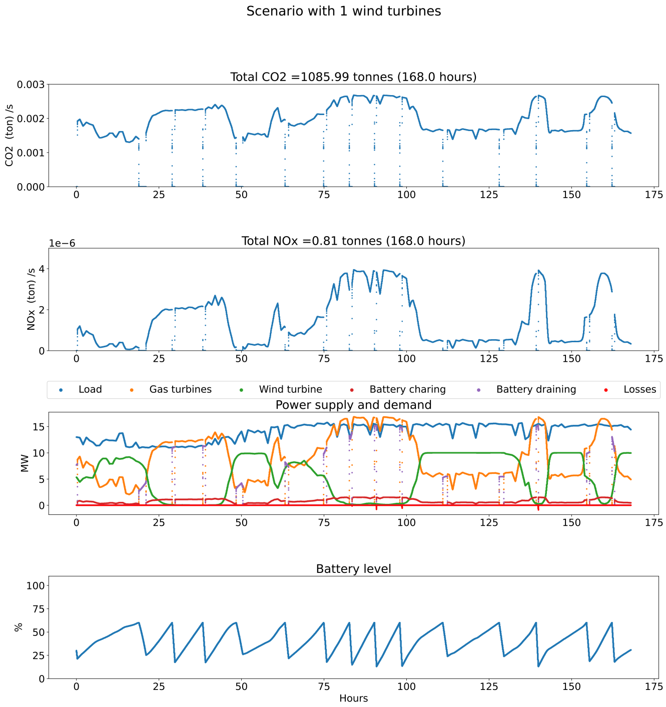


Figure 12. Simulation with 1 wind turbine

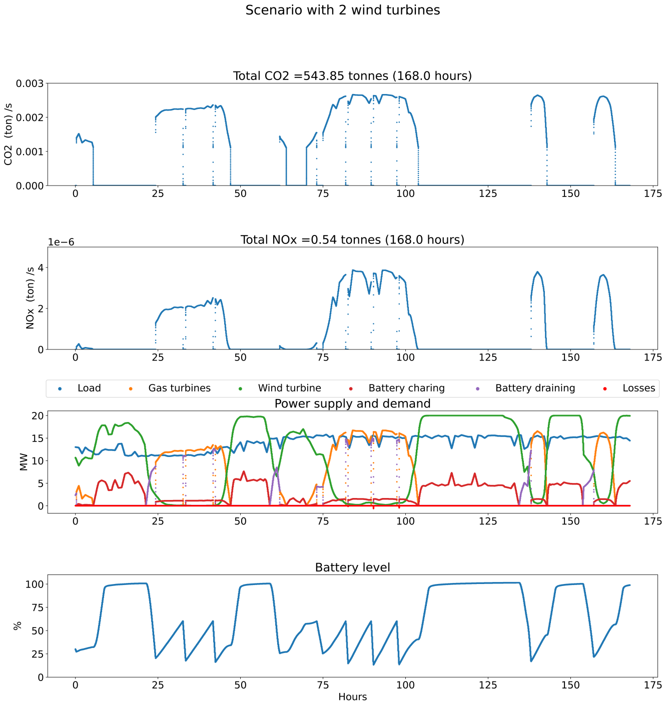


Figure 13. Simulation with 2 wind turbine

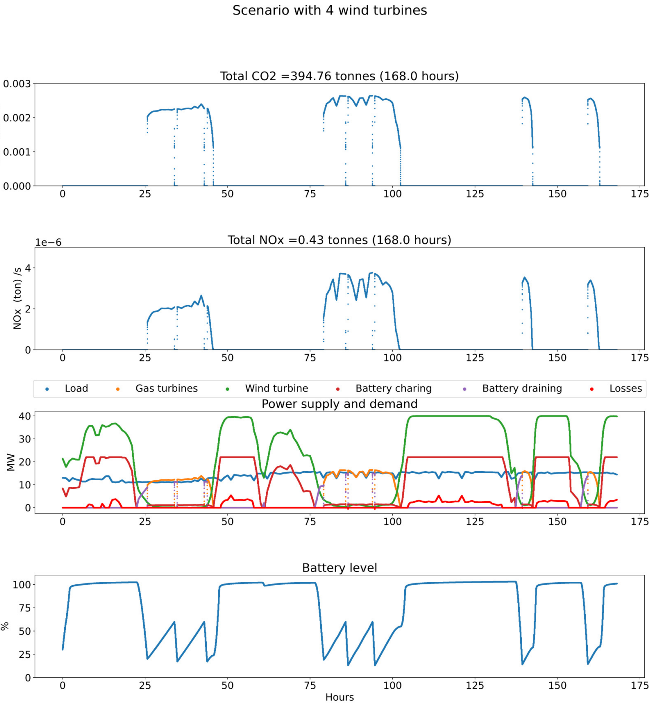


Figure 15. Simulation with 4 wind turbine

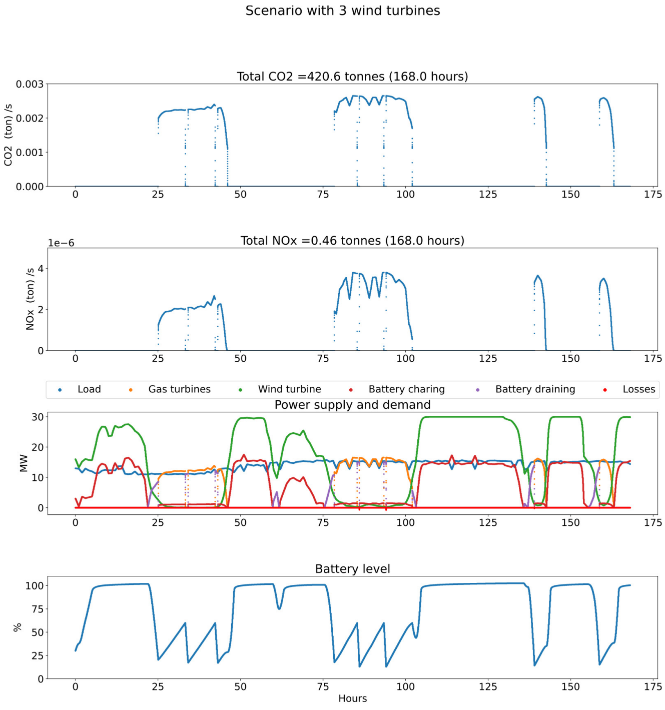


Figure 14. Simulation with 3 wind turbine

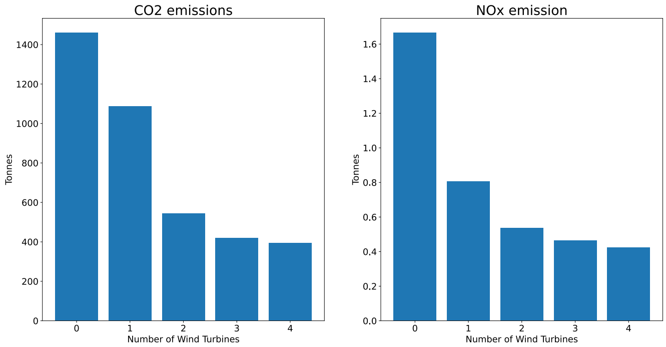


Figure 16. Summary of greenhouse gases emissions for 7 days simulation

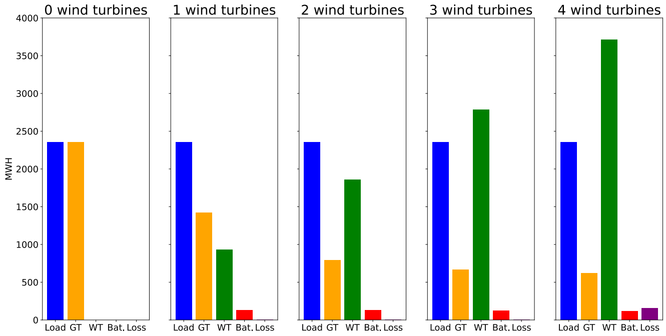


Figure 17. Power contribution for each scenario

4 Conclusions

In conclusion, this paper has presented a simple and yet useful modelica model (ELOGOW 2021) for wind power integration with gas turbines and energy storages to reduce greenhouse emission gases for offshore oil and gas platforms. The paper also presented a possible control strategy to autonomously keep the micro grid in balance. Furthermore, in this model we deliberately avoided detailed component modeling as a design principle because these can vary between installation making model reconfiguration challenging and maybe even unpractical. However, many aspects of the model can be improved on literally every component presented. Here are some suggestions.

4.1 Gas turbine model

The amount of gas emissions depends on many parameters often unknown or are particular to every installation. Having good quality datasets on the gas emission would make it possible to have data driven models with better predictions. The same argument also holds for the startup and operational phases of gas turbines.

4.2 Wind Turbines

When more than one wind turbine is involved, their placement becomes important to avoid the wake effect, which in often cases is inevitable. Improving the wind turbine model with a wind speed weakening estimation would provide better estimations of the generated power. A possible approach would be to apply a fast wind park simulator such as (NREL 2021).

4.3 Energy Storage System

Energy storage systems have different characteristics. In this paper we model them as energy stored, power released, and power consumed. Yet, the dynamic of such systems is expected to be more complex and could eventually be derived from data (test cases) in combination with more detailed models such as (Gerl et al. 2014). The optimal size of the storage capacity will also be explored in future studies. Limiting factors are the physical size, weight, and cost of the components.

4.4 Micro Grid

The micro grid should be extended to the electrical domain and should be made more realistic for operations by adding frequency drop and rise as a function of the imbalance between the power supply and demand.

4.5 Demand

We have considered one dataset representing the power demand of a typical oil platform in the North Sea. However, the demand can vary in many ways not considered in this paper. Having different datasets or eventually demand simulating models would improve the quality of the estimations.

4.6 Optimization

The control strategy presented in this paper should be formalized as a multi objective optimization problem. These objectives could be reduced greenhouse gas emissions, improved battery life cycle, dynamic triggering limits, improved gas turbines usage, etc.

Acknowledgements

The authors would like to acknowledge Equinor of providing the datasets and IFE for advice on energy storage sizes. This work has been performed as part of the project Electrification of Oil and Gas Installation by Offshore Wind (308838 - KSPKOMPETANSE19) financed by The Research Council of Norway.

References

Aguilera, Miguel et al. (2019-02). “Coalesced Gas Turbine and Power System Modeling and Simulation using Modelica”. In: DOI: 10.3384/ecp1815493.

Bak, Christian et al. (2013). *The DTU 10-MW Reference Wind Turbine*. Danish Wind Power Research 2013 ; Conference date: 27-05-2013 Through 28-05-2013.

Bosch, Jonathan, Iain Staffell, and Adam D Hawkes (2018). “Temporally explicit and spatially resolved global offshore wind energy potentials”. In: *Energy* 163, pp. 766–781. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2018.08.153. URL: http://www.sciencedirect.com/science/article/pii/S036054421831689X.

ELOGOW (2021). *https://github.com/NORCE-Energy/ELOGOW*.

ERA5 (2020). *www.ecmwf.int/en/forecasts/datasets/reanalysis-datasets/era5*.

Gerl, Johannes et al. (2014-03). “A Modelica Based Lithium Ion Battery Model”. In: DOI: 10.3384/ecp14096335.

He, Wei et al. (2010). “The Potential of Integrating Wind Power with Offshore Oil and Gas Platforms”. In: *Wind Engineering* 34 (2), pp. 125–137. DOI: 10.1260/0309-524X.34.2.125. URL: https://doi.org/10.1260/0309-524X.34.2.125.

NREL (2021). *https://github.com/NREL/floris*.

Pires, Thiago S. et al. (2018-04). “Application of nonlinear multivariable model predictive control to transient operation of a gas turbine and NOX emissions reduction”. In: *Energy* 149. ISSN: 03605442. DOI: 10.1016/j.energy.2018.02.042.

Robertson, A. et al. (2014-09). “Definition of the Semisubmersible Floating System for Phase II of OC4”. In: DOI: 10.2172/1155123. URL: https://www.osti.gov/biblio/1155123.

SSB (2020-11). *https://www.ssb.no/en/natur-og-miljo/statistikker/klimagassn*.

Zografos, Dimitrios, Mehrdad Ghandhari, and Robert Eriksson (2018-08). “Power system inertia estimation: Utilization of frequency and voltage response after a disturbance”. In: *Electric Power Systems Research* 161. ISSN: 03787796. DOI: 10.1016/j.epsr.2018.04.008.

Implementation and Validation of the Generic WECC Photovoltaics and Wind Turbine Generator Models in Modelica

Maria Nuschke[1] Sören Lohr[1] Adrien Guironnet[2] Marianne Saugier[2]

[1]Fraunhofer Institute for Energy Economics and Energy System Technology IEE, Power System Stability and Converter Technology Division, Germany, {maria.nuschke,soeren.lohr}@iee.fraunhofer.de
[2]Réseau de Transport d'Electricité, France, {adrien.guironnet, marianne.saugier}@rte-france.com

Abstract

This paper presents the open-source implementation in Modelica of the generic photovoltaics and wind turbine generator models introduced by the Western Electricity Coordinating Counsil (WECC) Renewable Energy Modeling Task Force. These dynamic models have been designed to be easily understandable and reusable by adopting the same decomposition as in the original WECC reports. It uses as much as possible existing Modelica Standard Library blocks and extends common parts whenever possible. The simulation results obtained with OpenModelica[1] and Dynaωo [2] - an hybrid C++/Modelica open source suite of simulation tools for power systems - have been successfully validated against different reference tools.

Keywords: Power System Modeling, Renewable Energy Sources, PV Models, Wind Turbine Generator Models, Open-Source

1 Introduction

Power system stability is challenged by increasing shares of Inverter-Based Generation (IBG) and systems operators' access to models representing the dynamic behavior of IBG realistically is fundamental in order to ensure a secure and safe network operation. On the other side, the exact implementation of the actual plant and generator control is treated as confidential by inverter manufacturers. Therefore, several efforts have been conducted in the past few years to propose generic or standard models for IBG. The large-scale Photovoltaics (PV) and Wind Turbine Generator (WTG) models proposed by the Western Electricity Coordinating Council (WECC) Renewable Energy Modeling Task Force (REMTF) (Ellis and et.al. 2012) are modular open-source models that enable users to perform stability studies while considering the dynamic behavior of most large PV farms and WTG installations realistically and independently from specific vendors.

Since the first definition of the generic WECC PV and WTG models in 2012, several implementations in commercial software environments have been presented, e.g. (Gustav Lammert, Luis David Pabon Ospina, et al. 2016) and various stability studies have been demonstrating the dynamic behaviour of these models under different conditions realistically, e.g. (G. Lammert, L. D. Pabon Ospina, and al. 2017; Gustav Lammert, Premm, et al. 2017; Luis David Pabon Ospina et al. 2018; Nuschke et al. 2019; L. D. Pabon Ospina and T. V. Cutsem 2020; L. D. Pabon Ospina and T. Cutsem 2020).

Further improvements to the models have also been applied. For instance, a new voltage sourced interface (Pourbeik 2018; Ramasubramanian et al. 2017) as an extension to the conventionally used current source interface has been proposed to improve the numerical stability of the simulation in situations with very high shares of IBG.

In the meantime, Modelica has gained a growing interest in the power system community. In addition to already existing efforts driven by first-hours Modelica enthusiasts in the Modelica.Electrical.QuasiStationary or PowerSystems libraries, the European projects Pegase and iTesla have boosted the use of Modelica in the power system community. They notably contribute to prove the language usability for power system modeling (Chieh, Panciatici, and J.Picard 2011) and affirm its interest for unambiguous models implementation (Vanfretti et al. 2013). Since these projects end, more and more power system stakeholders are using Modelica either for academic works (Gonzalez-Torres et al. 2019; Mirz et al. 2019; Qin et al. 2019; Masoom et al. 2020) or industrial use (Casella et al. 2016; Guironnet, Saugier, et al. 2018; Guironnet, Rosière, and Bureau 2021). They are attracted by the flexibility, usability and robustness of the language coupled with the progresses done in Modelica tools for large-scale simulations (Braun, Casella, and Bachmann 2017; Henningsson, Olsson, and Vanfretti 2019), and the creation and availability of generic and easy to adopt libraries such as the PowerGrids (Bartolini, Casella, and Guironnet 2019) or Dynaωo (Guironnet, Saugier, et al. 2018) ones.

One key aspect favorizing the spread of the Modelica language in the power system community is undoubtedly its easiness for the modeling of non conventional components and usefulness for stability and design studies of such components. Indeed, the flexibility and freedom offered in the models development - no constraint on the interfaces, possibility to mix block and equation-based approaches, etc. - but also in the test case creation - possi-

[1] https://www.openmodelica.org
[2] http://www.dynawo.org

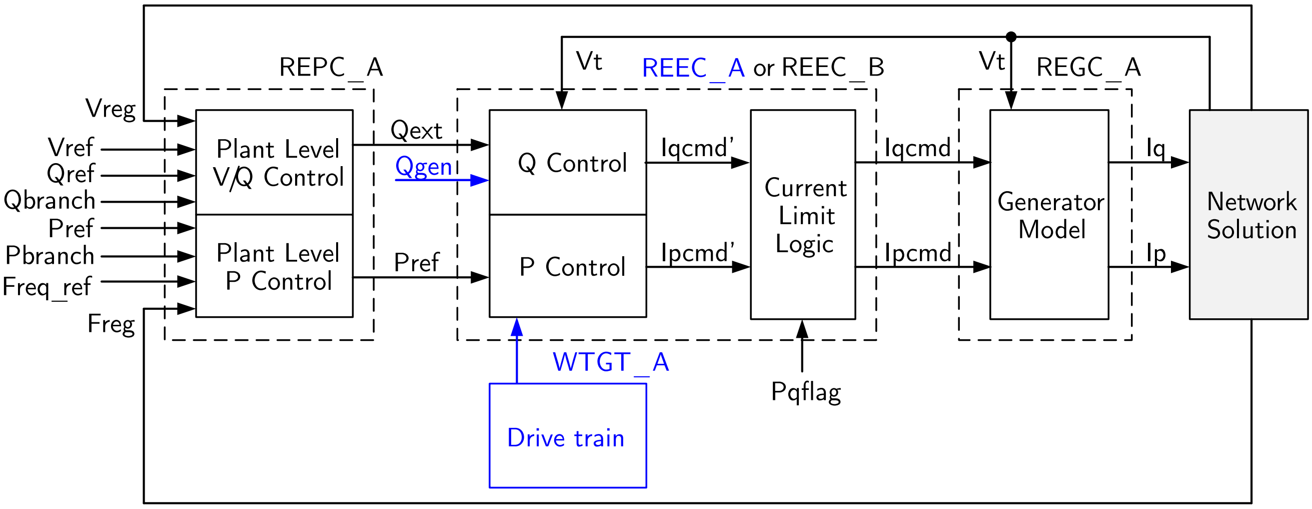


Figure 1. WECC block diagram PV (black), with additional blocks for WTG (blue), see Ellis and et.al. (2012)

bility to mix different kind of models for example - make it a good candidate for advanced studies. It is in particular the case in the field of power-electronics dominated systems as shown in (Cossart, Rosiere, et al. 2020). The release of different open-source components, e.g. (Murad, Gomez, and Vanfretti 2015; Cossart, Saugier, and Guironnet 2021), and use cases is one additional point to promote the use of the Modelica language in this direction.

The contribution of this paper is thus to present an implementation of the generic large-scale WECC PV and WTG models in Modelica and their validation against reference tools. To the best of the authors' knowdlege, it is the first open-source implementation of these models. Furthermore, their integration into the Dynaωo suite - the models are available in the Modelica library of the project [3] - enables to offer a wide variety of test cases in which these models can be integrated.

The rest of this paper is organized in the following way. Section 2 is devoted to the WECC PV and WTG models presentation while Section 3 describes their implementation. The design choices, the modeling approach and the implementation of a few selected components are detailed in this later section. Section 4 demonstrates the excellent level of accuracy obtained with the models by a comparison with reference results from a validation tool provided by EPRI as well as from the standard library object provided in DIgSILENT/PowerFactory. Finally Section 5 gives the conclusion.

2 Models Presentation

The WECC models and their updates are described in specific modeling guidelines and publications, e.g. (Ellis and et.al. 2012; Pourbeik 2018; Ramasubramanian et al. 2017). For convenience, the description of the most relevant blocks is given in the sequel.

2.1 Overview

The WECC PV and WTG type 4 (Type 4 is considered as fully rated converter WTG) models share a common high-level organization, as depicted in Figure 1 and are divided in three main control blocks:

- The plant control - called Renewable Energy Plant Control (REPC) - sets the main control choice for the whole plant. Voltage or reactive power control at plant level and frequency-dependent active power adjustment are part of the plant level. Identical in PV and WTG models.

- The electrical control - called Renewable Energy Electrical Control (REEC) - includes local inverter functionalities such as Fault-Ride Through (FRT) characteristic with fast reactive current injection, local voltage and reactive power control and current limitation with respect to the priority given to active or reactive current, respectively. There are two module versions available, whereas the REEC_B module is recommended for the WECC PV models and the REEC_A module is recommended for WECC WTG. The REEC_B module is a slightly simplified version from REEC_A module.

- The generator control - called Renewable Energy Generator Control (REGC) - is the last part of the control and interfaces with the grid. It enables to convert the current set-points calculated by the REEC part into the final currents (or voltages) delivered to the network. Identical in PV and WTG models.

For the WTG model a drive-train model - called Wind Turbine Generator Train (WTGT) - can be considered additionally in order to represent rotor speed changes and possibly resulting torsional oscillations after faults or sudden wind speed changes. By considering the drive-train model, the WTG model is equivalent to the WECC WTG

[3] https://github.com/dynawo/dynawo/tree/master/dynawo/sources/Models/Modelica/Dynawo

type 4A. By neglecting the drive-drain model, the rotor speed is considered to be constant at nominal value 1 p.u. and the WTG model is equivalent to the WECC WTG type 4B. The model types 4A and 4B are sub-types of the fully rated converter model for wind turbine generators.

2.2 Plant level control (REPC)

The plant level control determines the local inverter set-points for active power ($Pref$) and reactive power ($Qext$) as input to the REEC by considering measurements at Point of Common Coupling (PCC) and user-defined set-points at plant level.

The active power regulation is shown in Figure 2. Frequency dependent active power adjustment can be activated by setting the $FrqFlag$ to $true$ and is done by a proportional integer action on an addition of the frequency deviation and active power injection deviation. Note that if the adjustment is deactivated, $Pref$ will be directly passed to the REEC.

The reactive power regulation control is displayed in Figure 3. Reactive power control can be realized with reference either to reactive power or to voltage amplitude but is always done with a proportional integer action. The choice is made with the parameter $RefFlag$. Typically, in voltage control mode, the regulated bus is the point of common coupling and measurement values are available from that bus. The regulated bus can also be chosen different from the PCC. Therefore, the parameter $VcmpFlag$ has to be set to $true$ and the impedance $Rc + jXc$ specifies the impedance between the PCC and the desired remotely controlled bus.

2.3 Electrical control (REEC)

The electrical control is itself divided in two main functions:

- The first part determines from the input set-points $Pref$ and $Qext$ the necessary currents to inject in the network.
- The second part is the current limiter logic that will potentially limit the current injections through a certain process, that differs between the PV and WTG models.

In the P control, the currents direct component $ipcmd$ is calculated through a first-order structure with limits on P and its derivative and then on the current value itself. Note that the variation can be frozen by an external signal, in case of a voltage dip.

In the Q control, the overall structure is more complex and handles different kind of control modes. If a local coordinated V/Q control is activated on top of the plant control, there are different local proportional-integral actions applied on the input signal. Otherwise, the control is similar to the P section with a first-order structure. Both control loops (with or without local coordinated V/Q control) can be frozen by an external loop signal, in case of a voltage dip. Finally, the last part of the control structure - the upper part - corresponds to an additional current injection that is activated in FRT situations. All these loops are visible in Figure 4.

Regarding the current limiter logic, the priority between active and reactive support is defined through a flag. If the flag prioritizes the active current injection, the limits are defined in the following way:

$$ipmax = imax \tag{1}$$

$$ipmin = 0 \tag{2}$$

$$iqmax = \sqrt{(imax^2 - ipcmd^2)} \tag{3}$$

$$iqmin = -iqmax \tag{4}$$

Vice versa, in case of priority given to reactive current injection, the active current component will be reduced in favor of reactive current.

The reactive current injection during faults (Fault ride through capability, FRT) is implemented as a voltage dependent current injection for the PV model. For the WTG model, the FRT behavior is defined by three potential states, as depicted in Figure 5, where either no injection, voltage dependent injection or a constant injection is made. The parameter $Thld$ decides what happens after a voltage dip has ended.

2.4 Generator control (REGC)

The generator control REGC_A as per the initial implementation proposal from WECC (Ellis and et.al. 2012) calculates the set-points for active and reactive current considering ramp rate limiters for the currents or active power. The enhancements introduced in (Pourbeik 2018) including reference voltage calculation or even current control and phase-locked-loop are represented in different sub-modules REGC_B and REGC_c, respectively.Rotor

2.5 Control modes

As already shown in the previous parts, the models offer several degrees of freedom and enable to activate or deactivate different kinds of controls by changing the values of the corresponding flags. Table 1 illustrates this with a few examples that allow to model different strategies for voltage or reactive power control.

3 Modelica Implementation

3.1 Design choices

In the context of the WECC models and considering the models structure presented in the previous section, the implementation in Modelica has been done in the following way:

- The injector that connects the PV or WTG model to the network, as well as the voltage drop between the PCC and the voltage set-point or the current limitations logics are described through equations as specific sub-components.

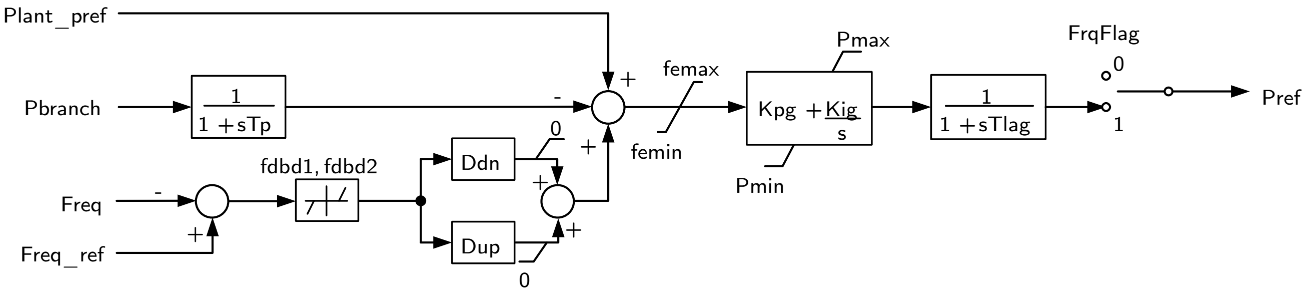


Figure 2. REPC - Active power control according to Ellis and et.al. (2012)

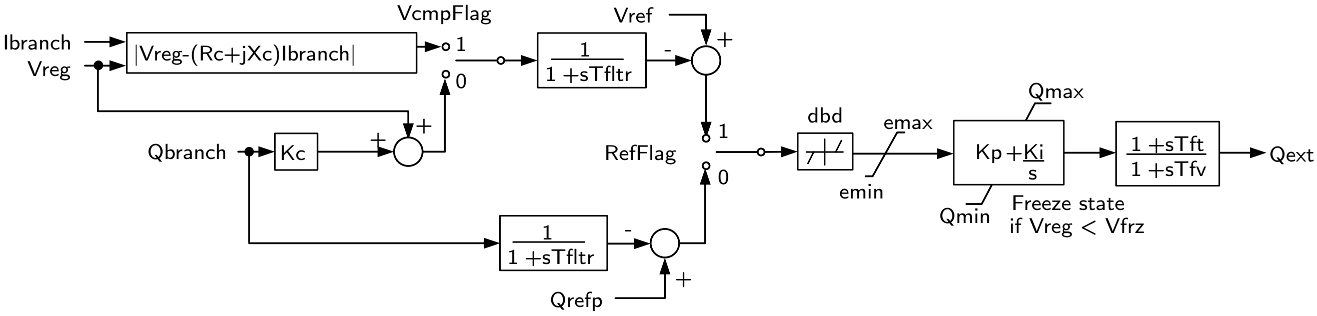


Figure 3. REPC - Reactive power control according to Ellis and et.al. (2012)

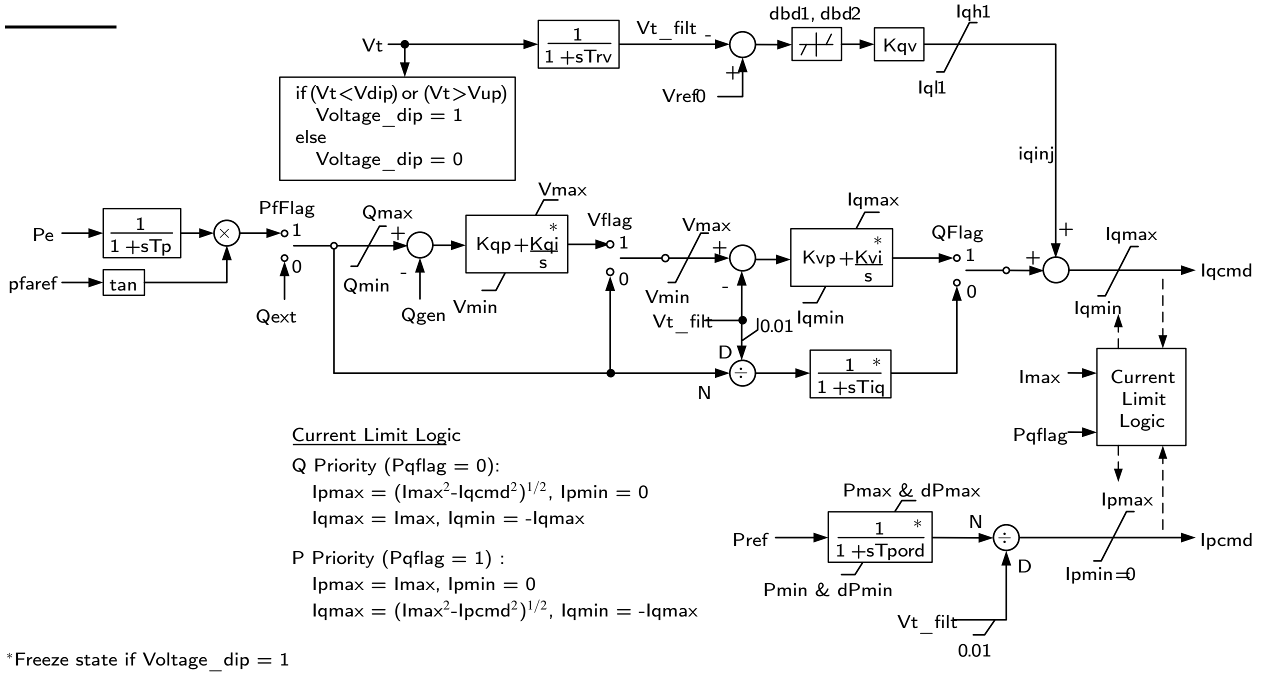


Figure 4. REEC_B - Electrical control according to Ellis and et.al. (2012)

- The other parts of the control are split in a similar way to the WECC original models, based on their behavior, and are described through a diagram approach.

The major parts of the models are thus built by combining individual and elementary blocks in a very similar way to the original WECC documentation (Ellis and et.al. 2012). In order to ease the long-term maintenance and to benefit from the robustness of widely used components, the Modelica Standard Library has been used as much as possible in the models.

Complementary blocks have been developed to handle specificities of the WECC models, such as the possibility to freeze the block actions with an external order sig-

Table 1. Control functionalities with plant level control and electrical control in service Ellis and et.al. (2012)

No.	Control mode	PFFlag	VFlag	QFlag	RefFlag
1	Plant level Q control	0	N/A	0	0
2	Plant level V control	0	N/A	0	1
3	Plant level Q control with local coordinated V/Q control	0	1	1	0
4	Plant level V control with local coordinated V/Q control	0	1	1	1

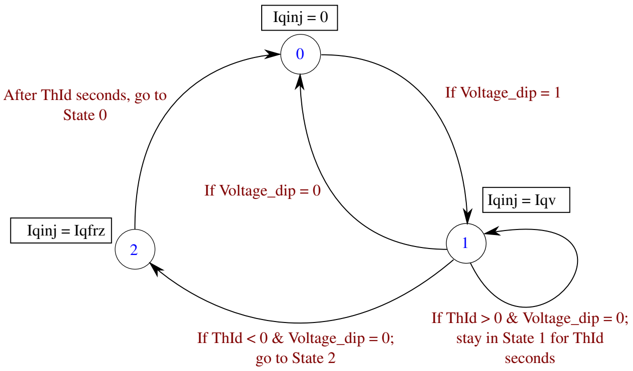

Iqinj = 0
0
After ThId seconds, go to State 0
If Voltage_dip = 1
If Voltage_dip = 0
Iqinj = Iqv
Iqinj = Iqfrz
2
1
If ThId > 0 & Voltage_dip = 0; stay in State 1 for ThId seconds
If ThId < 0 & Voltage_dip = 0; go to State 2

Figure 5. FRT behavior for the WECC WTG model according to Ellis and et.al. (2012)

nal. These additional blocks have been kept as generic as possible to facilitate their reuse through the whole models implementations.

3.2 Architecture

PV and WTG models have a very common structure, composed of identical REPC and REGC controls. Only part of the REEC control is different between the two generic models, and WTG model also has one additional structure to represent the drive-train behavior.

To minimize the number of models developed and reuse as much as possible common parts, the REPC and REGC controls have been developed only once. A common basis has also been defined for the REEC control, enabling to extend it to build the two final REEC control blocks REEC_B and REEC_A for PV and WTG models, respectively. It is worth mentioning that a large part of the block is implemented in the common basis. The same approach has been used for two possible implementations of the WTGT block.

Finally, the complete PV or WTG models are obtained by combining the different parts of the control with an injector providing the currents to the network. With such an architecture, modifying one part of the model or adding a new block is very easy and straightforward; it can be done without any modifications on the other parts.

3.3 Illustration

Figure 6 depicts the overall composite model, the similarity to Figure 1 is obvious. Further, detailed Modelica implementations of some WECC model blocks are given.

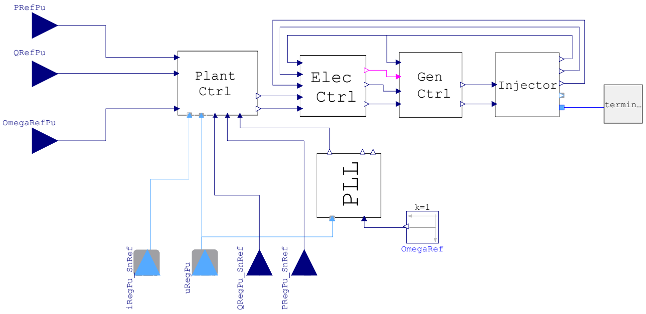


Figure 6. Block diagram of the WECC Large-scale PV model in Modelica, see Figure 1 for comparison.

3.3.1 Plant level control (REPC)

Figure 7 shows the implementation in Modelica of the REPC control: only the three blocks in blue have been specifically implemented for this control and are not part of the Modelica Standard Library. They represent the calculation of the voltage drop for the voltage control at a remote bus, the activation or deactivation of the freeze feature in case of voltage dip or increase and a modified implementation of a PID block to take into account the integrator state freeze.

The voltage drop calculation for voltage control at a remote busbar, is for example a short block containing the following equations:

```
uLineDrop = uPu + iPu * Complex(Rc,Xc);
UPuLineDrop = ComplexMath.'abs'(uLineDrop);
```

3.3.2 Electrical control (REEC)

Figure 8 depicts the Modelica implementation for the REEC of the WTG model. The parts surrounded in green are the ones specific for the WTG model while the others are inherited from the common control defined for both the PV and WTG models. A large part of the control is thus implemented only once in the base model.

The FRT logic shown in Figure 5 has been implemented using an algorithm approach enabling to handle the different transitions between states.

```
algorithm
  when Voltage_dip == true then
    Vdip_start := true;
    Vdip_inj_endTime := -1;
  end when;
  // Vdip has ended, set timing and reset Vdip
      detection variable
  when Voltage_dip == false and Vdip_start ==
      true then
```

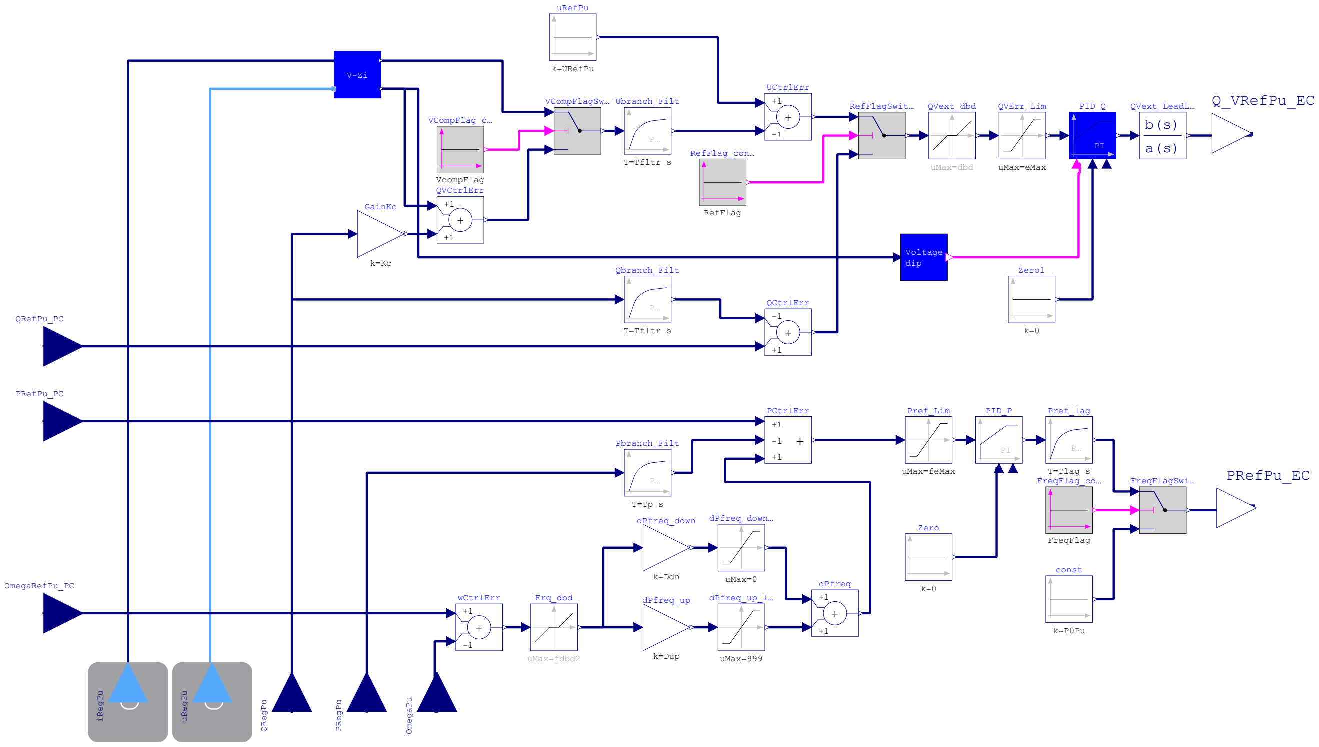


Figure 7. REPC - Park level control in Modelica, see Figures 2 and 3 for comparison.

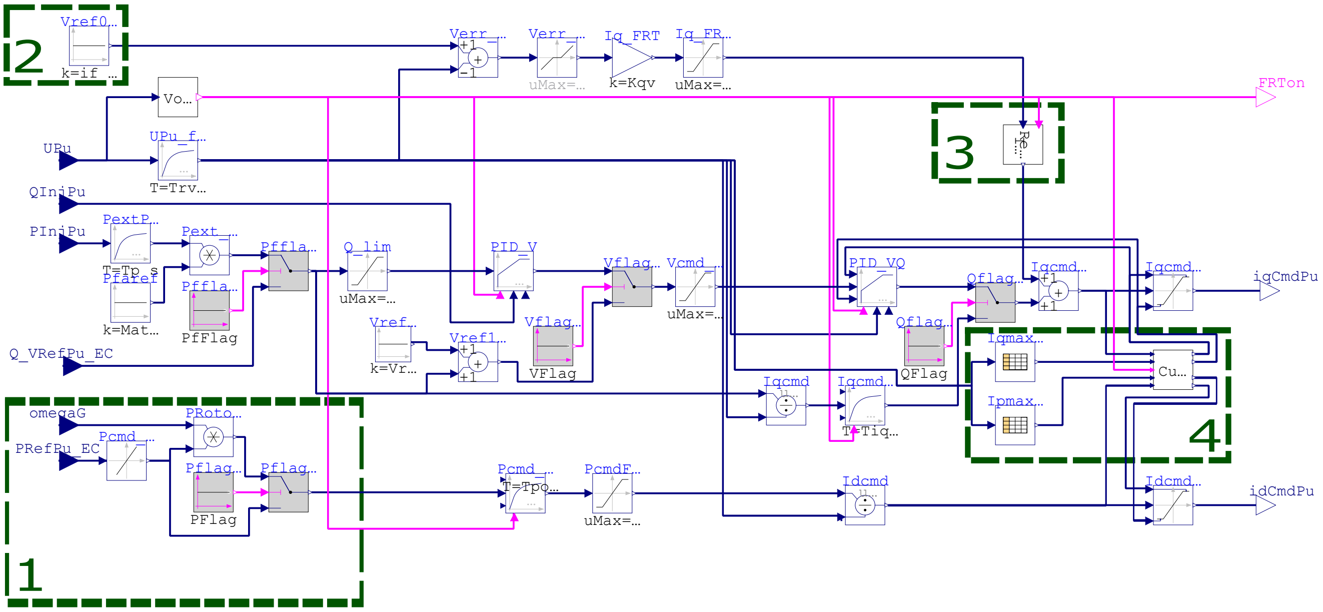


Figure 8. REEC - Electrical control in Modelica, green boxes highlight the specific blocks for the WECC WTG models: 1 - consideration of variable generator speed, 2 - additional voltage reference, 3 - extended FRT logic, 4 - extended current limitation logic, see Figure 4 for comparison.

```
    Vdip_start := false;
    Vdip_inj_endTime := time + abs(Thld);
  end when;
  // End time reached, reset.
  when time >= Vdip_inj_endTime and
      Vdip_inj_endTime >= 0 then
    Vdip_inj_endTime := -1;
  end when;
```

The current limit logic implementation for the WTG REEC model, that takes into account the existence of a voltage dip or not, is presented below. It is a mix between an algorithm part to detect the voltage dip and an if clause to choose the correct minimum and maximum values depending on priority given to active or reactive power.

```
algorithm
  when Voltage_dip == true then
    Vdip_start := true;
```

```
      Vdip_frz_endTime := -1;
    end when;
    when Voltage_dip == false and Vdip_start ==
        true then
      Vdip_start := false;
      Vdip_frz_endTime := time + abs(Thld2);
      Ipmax_frz := Ipmax;
    end when;
    when time >= Vdip_frz_endTime and
        Vdip_frz_endTime >= 0 then
      Vdip_frz_endTime := -1;
      Ipmax_frz := 0;
    end when;
  equation
    if PqFlag then
    // P priority
      if time <= Vdip_frz_endTime and
          Vdip_frz_endTime >= 0 then
        Ipmax = Ipmax_frz;
      else
        Ipmax = min(Ip_vdl, Imax);
      end if;
      Ipmin = 0;
      Iqmax = min(Iq_vdl, sqrt(Imax ^ 2 - min(
          Ip_lim,Ip_vdl) ^ 2));
      Iqmin = -Iqmax;
    else
    // Q priority
      if time <= Vdip_frz_endTime and
          Vdip_frz_endTime >= 0 then
        Ipmax = Ipmax_frz;
      else
        Ipmax = min(Ip_vdl, sqrt(Imax ^ 2 - min(
            Iq_lim,Iq_vdl) ^ 2));
      end if;
      Ipmin = 0;
      Iqmax = min(Iq_vdl, Imax);
      Iqmin = -Iqmax;
    end if;
```

3.3.3 Wind generator turbine drive-train (WTGT)

The drive train model is a simplified model for the purpose of emulating the behavior of torsional mode oscillations. The shaft damping coefficient ($Dshaft$) in the drive-train model is fitted to capture the net damping of the torsional mode seen in the post fault electrical power response. The mechanical power *Pm* is initialized with the initial value of the electrical power *Pe*. For this implementation proposal in Modelica, the block diagram approach has been chosen in order to follow the original WECC reference implementation. It is also possible to use either physical Modelica components or go for an equation-based modeling.

3.3.4 Current injector

In the initial model proposed by the WECC, the interface to the network is a current source, therefore a current injector element has been implemented in Modelica. It means that the REGC control calculates the d-axis and q-axis currents that are then converted to the actual currents injected to the network by a current injector using the Park transformation:

```
terminal.i.re = -(cos(UPhase) * idPu - sin(
    UPhase) * iqPu) * (SNom/SystemBase.SnRef);
terminal.i.im = -(sin(UPhase) * idPu + cos(
    UPhase) * iqPu) * (SNom/SystemBase.SnRef);
```

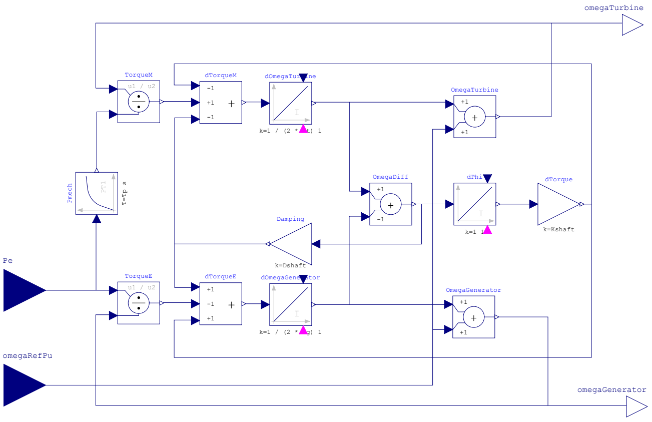


Figure 9. WTGT - Drive train model in Modelica

3.3.5 Voltage injector

Another interface based on directly imposing the voltage at the PCC has been introduced by the WECC, notably for higher numerical stability with very high shares of IBG. In this case, an additional block is added to convert the d-axis and q-axis currents into inner real and imaginary voltage set-points that are then imposed to the network by a simple voltage injector, that sets the real and imaginary terminal voltages directly, see Figure 10.

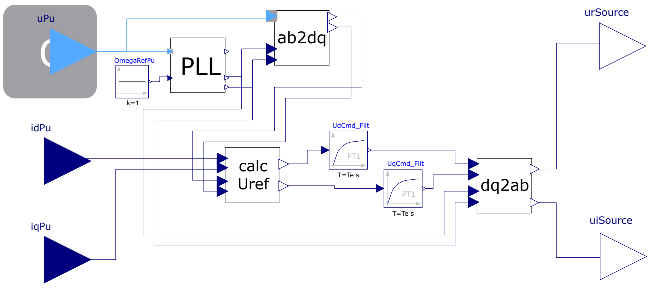


Figure 10. Voltage source reference in Modelica

The calculation of the reference voltage in the dq-reference frame is implemented with the following equations using the inverter impedance $R+jX$:

```
ud_ref = ud + id_ref * R - iq_ref * X;
uq_ref = uq + iq_ref * R + id_ref * X;
```

4 Validation

The models were validated by using the "Renewable Energy Model Validation Tool" (REMVT), written by EPRI, and additionally against a standard library implementation in DIgSILENT PowerFactory 2019, SP3. Notice that the REMVT itself was validated against real measurements.

The following sections briefly describe the test system used and then present the validation results by comparing simulation results from OpenModelica Connection Editor (OMEdit v1.13.2) with the results obtained from REMVT (Version 2.1) and the from DIgSILENT PowerFactory 2019 using the standard library object of the

WECC PV model (DIgSILENT/PowerFactory 2019).

4.1 Test case

The following single line diagram shows the considered test system consisting of the IBG model, a step-up transformer, equivalent park lines and the grid-connection transformer (substation) connected to an infinite bus or external grid. The HV side of the substation (bus 4) is considered as PCC. The plant level control regulates active power, reactive power/voltage at PCC. For unambiguous voltage control at plant level, the PCC and the infinite bus must be separate buses.

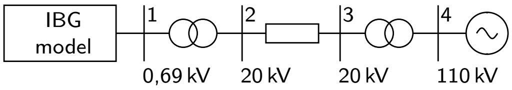


Figure 11. Single line diagram of the test system

Parameters for the equipment have been set according to Table 2. The conversion ratio for the transformers is set to one. Per unit base is $SnRef = 100\,\text{MVA}$.

Table 2. Test system parameters for validation

Component	Parameter	Value in p.u.
Step-up transformer, 20 kV pu Base	R	0.000100
	X	0.049999
	B	-0.004999
	G	0.000110
Line 20 kV pu Base	R	0
	X	0.000025
	B	0
	G	0
Substation transformer, 20 kV pu Base	R	0.000550
	X	0.099999
	B	-0.004999
	G	0.000020

The test scenario consists of a voltage dip of 50 % at $t = 1..2$ s and a frequency step up to 1.01 p.u. at $t = 6..9$ s at the infinite bus.

4.2 Validation results - PV model

This section exemplarily presents validation results from the WECC PV model with the conventional current sourced interface against the EPRI tool REMVT or the reference implementation in DIgSILENT/PowerFactory, respectively.

Figure 13 displays the validation results for control mode Nr. 1, (refering to table 1) plant level reactive power control. Simulation begins in steady state condition according to the given set-points. With the voltage dip at $t = 1..2$ s, reactive power increases due to fast reactive current injection from FRT mode, at the same time the active power is reduced due to the current limitation and priority given to reactive current. After voltage recovery, active and reactive power are controlled to reach the initially given set-points again.

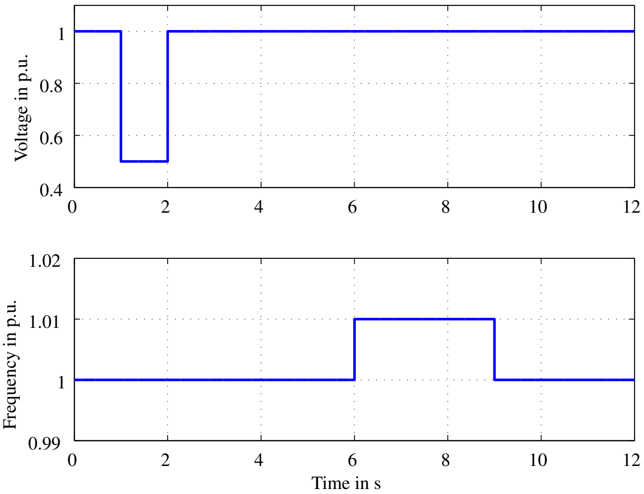


Figure 12. Test events: voltage magnitude and frequency at infinite bus terminal.

At $t = 6$ s the active power is reduced due to the increase in frequency with respect to the given droop $Ddn = 20$ p.u. (pu base SNom/fNom) per frequency deviation in *perunit*. As the frequency at infinite bus changes back to the nominal value, active power reaches the set-point as per the initially given value.

The results from REMVT and Modelica fit very well.

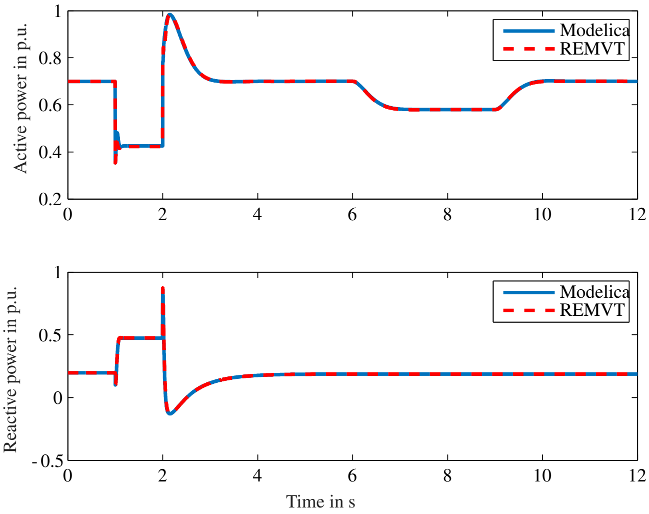


Figure 13. Simulation results PV model: active and reactive power in response to test events with control mode Nr. 1

Figure 14 displays the validation results for control mode Nr. 4, plant level voltage control with local coordinated V/Q control. The active power behavior is not affected by changing the control mode from Nr. 1 to Nr. 4 and the results from REMVT and Modelica are a good fit.

Reactive power behaves similar during the voltage dip because of the still active fast reactive current injection.

After voltage recovery, the reactive power settles slowly at a new steady state value according to the output of the plant level reactive power/voltage controller. The new steady state value is higher than the initial one because of the voltage dip: in order to support voltage, reactive power injection was increased by the plant level controller. Results from Modelica and REMVT are very well matching.

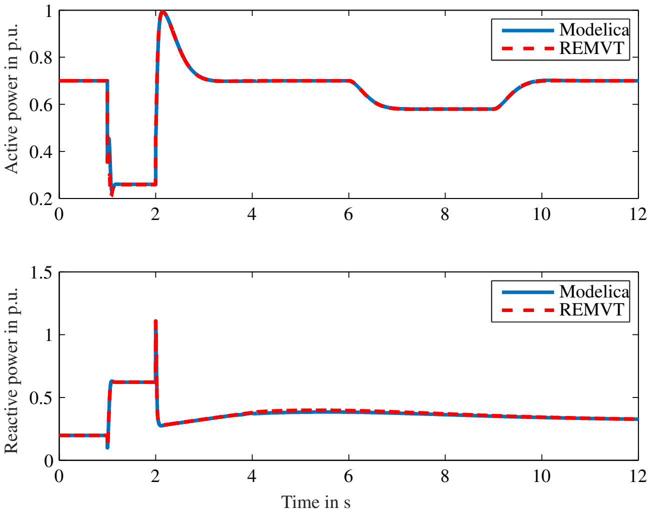


Figure 14. Simulation results PV model: active and reactive power in response to test events with control mode Nr. 4

As also the voltage source interface introduced in (Pourbeik 2018) has been implemented in Modelica, simulation results obtained from OpenModelica have been compared to the reference simulation tools as well.

Currently, the REMVT version does not contain a voltage source network interface. Therefore, if comparing the simulation results of the voltage source interfaced PV model obtained from OpenModelica against the results from REMVT with standard current source interface, the transient results are deviating, but steady state results fit very well. For validation against the reference implementation in PowerFactory, the voltage source interface was available and has therefore been used for validation.

Figure 15 shows exemplarily the validation results from OpenModelica against the reference implementation in PowerFactory for control mode Nr. 1, while the models in both simulation environments make use of the voltage source network interface. The results from both simulations environments match perfectly.

4.3 Validation results - WTG 4A model

Since the PV and WTG models share large portions of the control structure, the general behaviour of the WTG model without consideration of torsional oscillations (WTG 4B model) and the PV model is similar, therefore only validation results for the WTG 4A model including the drive train representation are presented. Figure 16 shows the validation results for control mode Nr. 4. The results from the Modelica implementation and the validation tool are exactly matching. Also the torsional oscillations in active power can be observed.

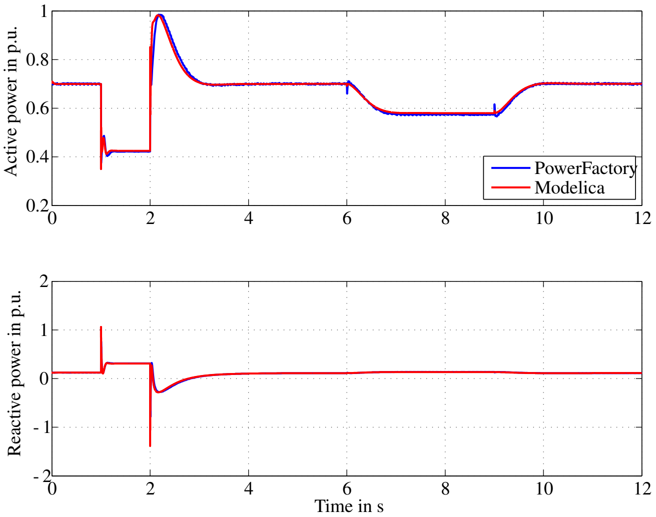


Figure 15. Simulation results PV model: active and reactive power in response to test events with control mode Nr. 1, Voltage source interface

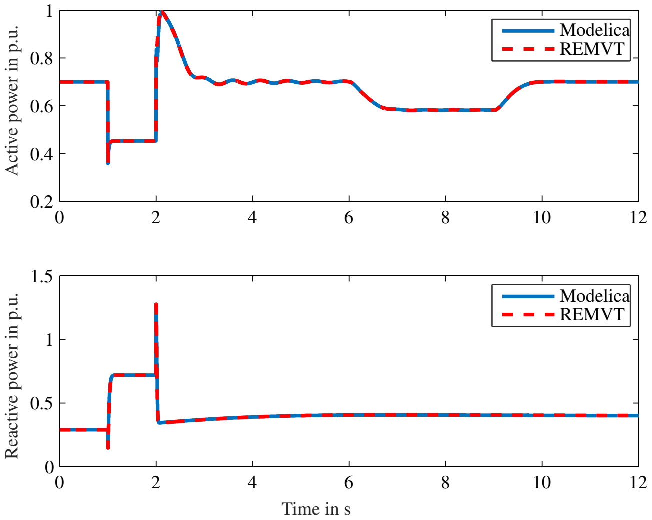


Figure 16. Simulation results WTG 4A model: Active and reactive power response with control mode Nr. 4.

5 Conclusion

This paper has presented an open-source implementation of the generic WECC PV and WTG models and their validation against both a validation tool provided by EPRI and an standard library implementation in DIgSILENT/PowerFactory. It demonstrates that the implementation in Modelica of such models is straightforward, easy to understand and to modify thanks to the native properties of the language (declarative and high-level language) and enables to achieve similar accuracy compared to traditional power system simulation tools. It also confirms that Modelica is an appropriate candidate for power system modeling and that its flexibility is a key feature for easy modeling of power-electronics dominated grids.

In the future, the authors plan to continue their work to both develop open-source Modelica models for standard and advanced power system components and make available standard test cases in order to encourage Modelica use in the power system community. Such an evolution will definitely facilitate technical discussions between all the power system stakeholders, from academics to industrials, and paves the way for a better and more coordinated handling of the numerous challenges arising in power system.

References

Bartolini, A., F. Casella, and A. Guironnet (2019-02). “Towards Pan-European Power Grid Modelling in Modelica: Design Principles and a Prototype for a Reference Power System Library”. In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. Linköing University Electronic Press.

Braun, W., F. Casella, and B. Bachmann (2017-05). “Solving large-scale Modelica models: new approaches and experimental results using OpenModelica”. In: *Proc. 12th International Modelica Conference*. Prague, Czech Republic, pp. 557–563.

Casella, F. et al. (2016-10). “Object-Oriented Modelling and Simulation of Large-Scale Electrical Power Systems using Modelica: a First Feasibility Study”. In: *Proceedings of the 42nd Annual Conference of the IEEE Industrial Electronics Society IECON 2016*. IEEE. Firenze, Italy: IEEE.

Chieh, A., P. Panciatici, and J.Picard (2011-06). “Power system modeling in Modelica for time-domain simulation”. In: *Proc. PowerTech*. IEEE.

Cossart, Q., F. Rosiere, et al. (2020-10). “An Open-Source Implementation of Grid-Forming Converters Using Modelica”. In: *2020 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)*. IEEE.

Cossart, Q., M. Saugier, and A. Guironnet (2021). *An Open-Source Average HVDC Model for Stability Studies*. Paper accepted for the 2021 IEEE PowerTech conference.

DIgSILENT/PowerFactory (2019). *Template Documentation WECC PV Power Plant Models (PV Models), Revision 1*. Tech. rep. DIgSILENT GmbH.

Ellis, E. and P. Pourbeik et.al. (2012). *Generic Solar Photovoltaic System Dynamic Simulation Model Specification*. Tech. rep. Western Electricity Coordinating Council (WECC) Renewable Energy Modeling Task Force.

Gonzalez-Torres, J.C. et al. (2019). “Power system stability enhancement via VSC-HVDC control using remote signals: application on the Nordic 44-bus test system”. In: *15th IET International Conference on AC and DC Power Transmission (ACDC 2019)*. Institution of Engineering and Technology.

Guironnet, A., F. Rosière, and G. Bureau (2021). *Dynaωo: A Suite of Power System Simulation Tools using Modelica and the OpenModelica Compiler*. Presentation done at the 2021 OpenModelica virtual Workshop. URL: https://openmodelica.org/events/openmodelica-workshop/openmodelica-program-2021.

Guironnet, A., M. Saugier, et al. (2018-10). “Towards an Open-Source Solution using Modelica for Time-Domain Simulation of Power Systems”. In: *2018 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*. IEEE.

Henningsson, E., H. Olsson, and L. Vanfretti (2019-02). “DAE Solvers for Large-Scale Hybrid Models”. In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. Linköing University Electronic Press.

Lammert, G., L. D. Pabon Ospina, and et al. (2017). “Impact of fault ride-through and dynamic reactive power support of photovoltaic systems on short-term voltage stability”. In: *IEEE Manchester PowerTech, Manchester*. IEEE.

Lammert, Gustav, Luis David Pabon Ospina, et al. (2016-07). “Implementation and validation of WECC generic photovoltaic system models in DIgSILENT PowerFactory”. In: *2016 IEEE Power and Energy Society General Meeting (PESGM), Boston, MA, USA*. IEEE. DOI: 10.1109/PESGM.2016.7741608.

Lammert, Gustav, Daniel Premm, et al. (2017-03). “Control of Photovoltaic Systems for Enhanced Short-Term Voltage Stability and Recovery”. In: *IEEE Transactions on Energy Conversion* 34, pp. 243–254. DOI: 10.1109/TEC.2018.2875303.

Masoom, A. et al. (2020-12). “Simulation of electromagnetic transients with Modelica, accuracy and performance assessment for transmission line models”. In: *Electric Power Systems Research* 189, p. 106799.

Mirz, M. et al. (2019-07). “DPsim—A dynamic phasor real-time simulator for power systems”. In: *SoftwareX* 10, p. 100253.

Murad, M.A.A., F. J. Gomez, and L. Vanfretti (2015-06). “Equation-based modeling of FACTS using Modelica”. In: *2015 IEEE Eindhoven PowerTech*. IEEE.

Nuschke, M. et al. (2019-09). “Power system stability analysis for system-split situations with increasing shares of inverter based generation”. In: *NEIS conference Hamburg*. IEEE.

Pabon Ospina, L. D. and T. Van Cutsem (2020). “Emergency support of transmission voltages by active distribution networks: a non-intrusive scheme”. In: *IEEE Transactions on Power Systems*. DOI: 10.1109/TPWRS.2020.3027949.

Pabon Ospina, L. D. and T.Van Cutsem (2020). “Power factor improvement by active distribution networks during voltage emergency situations”. In: *Elsevier Electric Power Systems Research Journal* 189.

Pabon Ospina, Luis David et al. (2018). “Impact of Plant-Level Voltage Control of Large-Scale Inverter Based Generators on Long-Term Voltage Stability”. In: *Power Systems Computation Conference (PSCC), Dublin*. IEEE. DOI: 10.23919/PSCC.2018.8442740.

Pourbeik, P. (2018). *Proposal for new features for the renewable energy system generic models, 07/23/18, latest revised 3/5/19*. Tech. rep.

Qin, Y. et al. (2019-08). “A JModelica.org Library for Power Grid Dynamic Simulation with Wind Turbine Control”. In: *2019 IEEE Power & Energy Society General Meeting (PESGM)*. IEEE.

Ramasubramanian, D. et al. (2017). “Converter Model for Representing Converter Interfaced Generation in Large Scale Grid Simulations”. In: *IEEE Transactions on Power Systems* 32.

Vanfretti, L. et al. (2013). “Unambiguous power system dynamic modeling and simulation using modelica tools”. In: *2013 IEEE Power & Energy Society General Meeting*. IEEE.

Modeling of Recompression Brayton Cycle And CSP Plant Architectures for Estimation of Performance & Efficiency

Ashok Kumar Ravi[1] Stéphane Velut[2] Raja Vignesh Srinivasan[3]

[1]Modelon Engineering Private Limited, India, ashokkumar.ravi@modelon.com
[2]Modelon AB, Sweden, stephane.velut@modelon.com
[3]Modelon Engineering Private Limited, India, rajavignesh.srinivasan@modelon.com

Abstract

As the world is moving towards renewable energy sources for sustainable energy, concentrated solar power (CSP) systems with thermal energy storage present significant opportunities for generating electricity. This paper describes an effort to develop an analytic platform for Brayton cycle and connect it to central receiver CSP system to form a complete system. Already analytical model for central receiver CSP system with Rankine cycle was already developed and available with Modelon (Edman, 2015; Windahl, 2015). This paper describes the development of a supercritical CO_2 recompression Brayton cycle based on the information available in the literature (Dennis, 2017). The effect of change in turbine inlet temperature on the performance and efficiency of Brayton cycle are shown. Integration of Brayton cycle with CSP system is done and the solar power requirement based on turbine inlet temperatures is studied.

Keywords: CSP, molten salt, Brayton cycle

1 Introduction

Due to the climate challenges arising because of heavy consumption of fossil fuels, lot of research is continuously being done on improving the efficiency of obtaining power through various renewable energy sources.

Concentrated solar power systems use mirrors to concentrate the solar energy. This concentrated solar power is transferred to the Molten salt and with this thermal energy storage through molten salt provides opportunity to utilize the energy during day or night as per power demand. This energy can be used in a steam generator or for Brayton cycle to generate electricity. Brayton cycle represents the operation of gas turbine engine. There is no phase change for the working fluid and fluid always remains in gaseous phase which is the reason for choosing supercritical CO_2 as the working fluid. Brayton cycle provides economic advantages such as smaller system size, increased efficiency, and environmental advantages such as greenhouse gas reduction, reduced water consumption etc.

Recompression Brayton cycle which requires additional compressor and heat exchanger (HX) is better than normal Brayton cycle due to its increased efficiency. Advanced control strategies can also lead to additional benefits by optimizing the heat flow from external fluid to supercritical CO_2, optimal operation of turbine and compressors under varying operating conditions. While careful design of the system is required to fully realize the benefits of Brayton cycle, there is clearly motivation to pursue given its economic and environmental benefits.

System modeling with Modelica language has been widely used for thermal power cycles modeling. With a powerful and flexible modeling framework and proven commercial libraries, Modelica language provides an ideal platform for architectural studies and controls prototyping for different power cycles. This paper describes an effort to develop an analytic platform for Brayton cycle and evaluate its performance under different operating conditions.

A baseline model of the system is developed with Vapor Cycle Library (Modelon AB, 2020). Results from the simulations are compared with data available in literature (Dennis, 2017). CSP system with thermal storage available in Thermal Power Library (Modelon AB, 2020) is modified such that Rankine cycle or Brayton cycle can be interchanged as the working cycle. Simulations are done in similar conditions using Rankine cycle and Brayton cycle to show the advantages of modular structure. Eventually this will help the user in evaluating the CSP system with Rankine and Brayton cycles and decision making.

2 Brayton Cycle System Model

This section provides an overview of the Brayton cycle model. The model is based on the prototype of recompression Brayton cycle developed by National Energy Technology Laboratory (NETL) (Dennis, 2017). The following sections provide an overview of the full system model and relevant component modeling details.

2.1 System and Model Overview

Brayton cycle system can be divided into low pressure and high-pressure sides as that of the Vapor cycle systems. From turbine outlet to the compressor inlet can be considered as low-pressure side, and from compressors outlet to the turbine inlet can be considered as high-pressure side. Following are the main components of the system:

- Turbine
- Main compressor
- Bypass compressor
- High temperature recuperator
- Low temperature recuperator
- Primary heater
- Cooler

Two variants of primary heaters are used based on the study. When integrated with CSP thermal storage system, counterflow heat exchanger is used with super critical CO_2 and Molten salt (60% $NaNO_3$ 40% KNO_3) as the fluids (two phase liquid heat exchanger). During individual system studies of Brayton cycle heat is directly provided to supercritical CO2. Two variants of the system are shown in Figure 3 and Figure 4 respectively. Conditional routing which is possible with Modelica language is used to pass the fluid through heat exchanger or volume with heat source based on requirement. In the first variant, volume with heat source is switched off and flow is happening through the heat exchanger (primary heater) only which is shown in red box in Figure 3. When the heat exchanger sizing and performance details are available this variant can be used. In the second variant, heat exchanger component is switched off and flow is passing through volume with heat source shown in red box in Figure 4. When the user has not sized the heat exchanger and want to calculate the amount of heat transfer required for the fluid to reach different target turbine inlet temperatures, this variant is very useful.

High temperature recuperator (HTR) and low temperature recuperator (LTR) are used for recovering the heat of the fluid coming out of the turbine. This increases the thermal efficiency of the system. In HTR, turbine outlet fluid interacts with the high-pressure fluid mixture coming out of bypass compressor and the secondary side of LTR. In LTR, fluid coming out of the HTR interacts with the fluid coming out of the main compressor. Fluid coming out of the LTR primary side is divided and sent through main and bypass compressors.

Flow through the bypass compressor is set such that more efficient heat recuperation happens, thereby higher cycle efficiency is achieved. Before sending the flow through the first compressor, fluid is cooled down to 35 ^{0}C. In the current model heat is removed directly using controller such that the inlet temperature at the main compressor inlet is 35 ^{0}C. Fluid inlet conditions to the bypass compressor are same as that of the exit conditions of the LTR.

After passing through the compressors and then the recuperators, high pressure fluid is provided with more heat such that the required electric power is generated.

2.2 System Characterization

One of the main challenges in building the model of the Brayton cycle system is the lack of data to parameterize the components and characterize the system outside of publicly available data in literature. This data along with knowledge of similar systems was used to get a reasonable, first cut system model though is admittedly imperfect and not desirable for model accuracy.

Compressors, turbine, and recuperators are the key components in the system. Dynamic compressor model based on look-up tables for isentropic efficiency and corrected mass flow rate is used for modeling the compressor. As we have one set of operating condition for the entire system based on literature (Dennis, 2017), that point is considered as the design point. Considering this as design point isentropic efficiency and mass flow rate maps (Figure 1, Figure 2, Figure 5, Figure 6) are generated which are shown below:

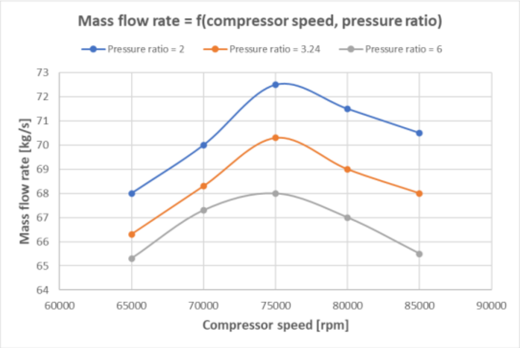


Figure 1. Main compressor mass flow map

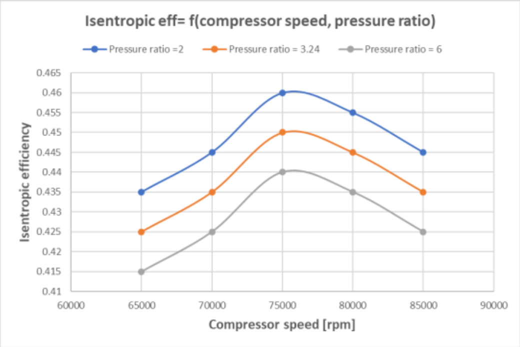


Figure 2. Main compressor isentropic eff map

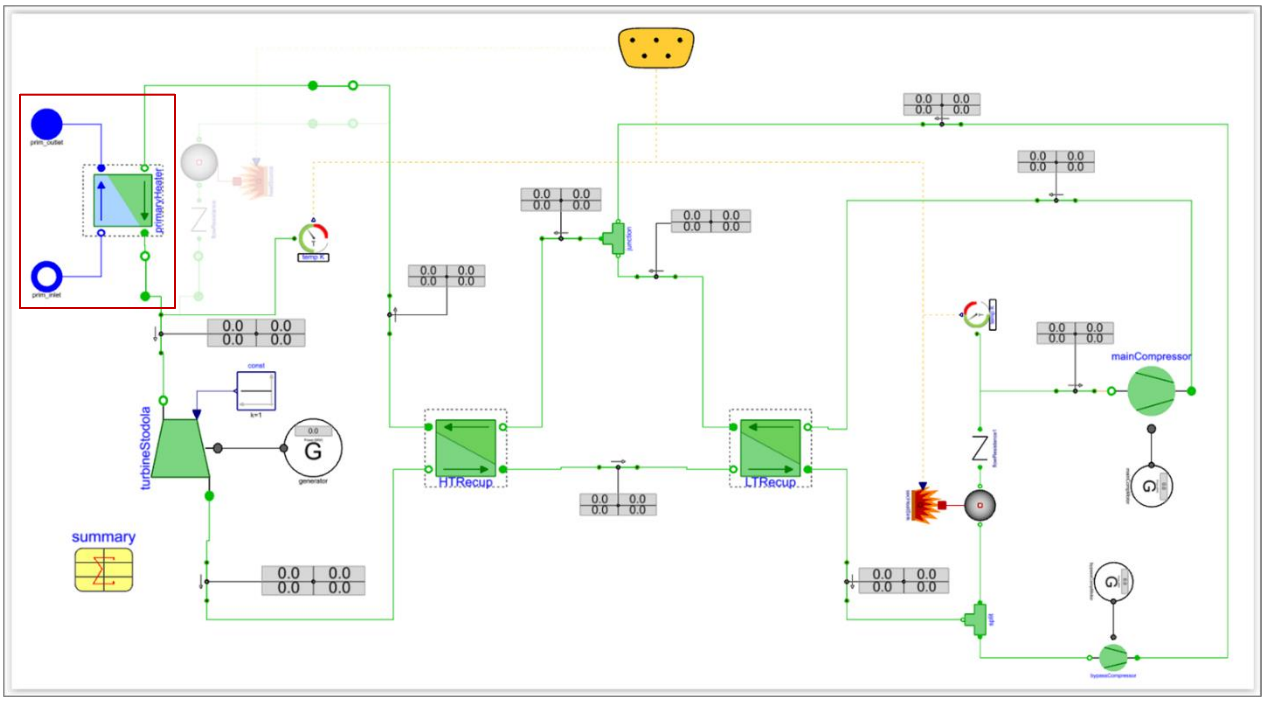


Figure 3. Brayton cycle with Heat exchanger as primary heater

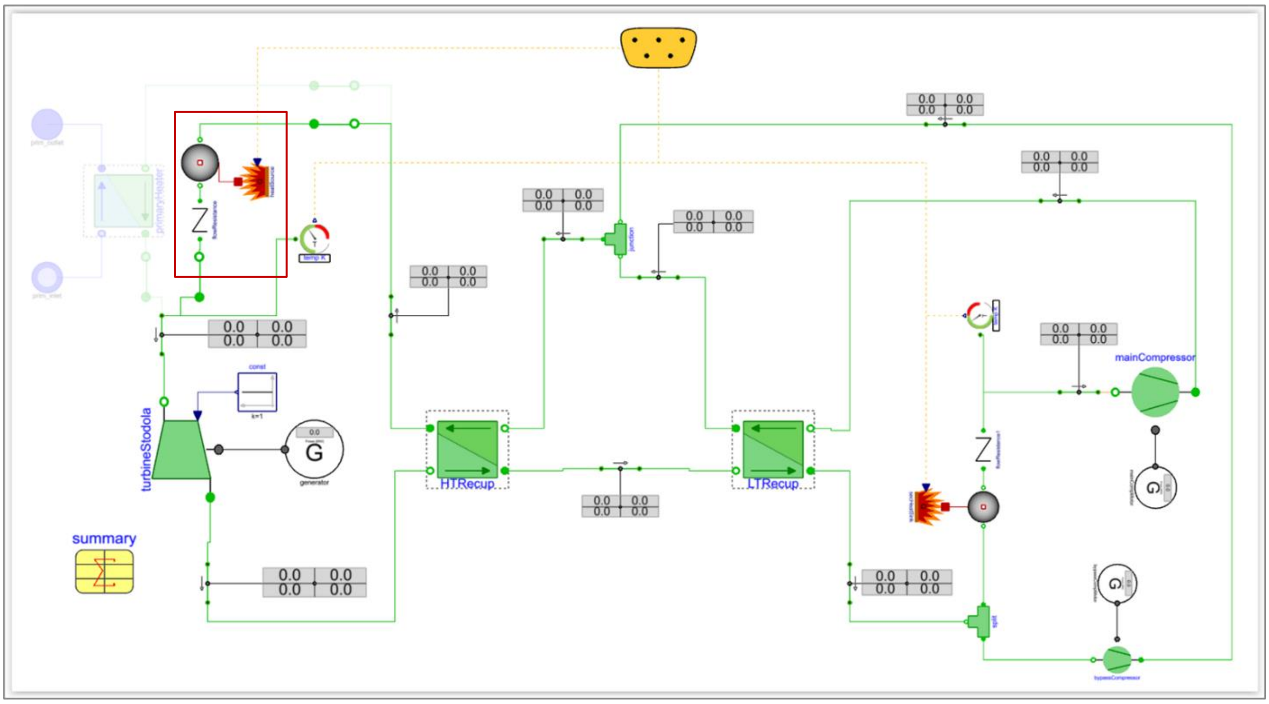


Figure 4. Brayton cycle with direct heat transfer to supercritical CO_2

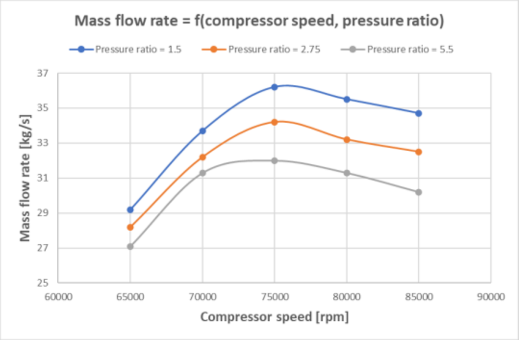


Figure 5. Bypass compressor mass flow map

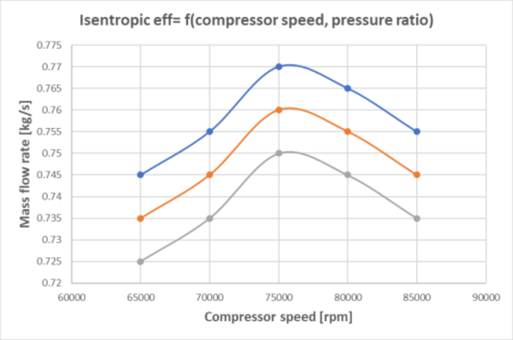


Figure 6. Bypass compressor isentropic eff map

Turbine model with flow according to Stodola's law is used with constant isentropic efficiency of 85 %. Generic counterflow two phase– two phase heat exchanger is used for modeling HTR and LTR. This heat exchanger can handle both single and two-phase refrigerant. Test benches are created for both the HX's and calibrated for the design point available in literature (Dennis, 2017). HTR test bench is shown below (Figure 7):

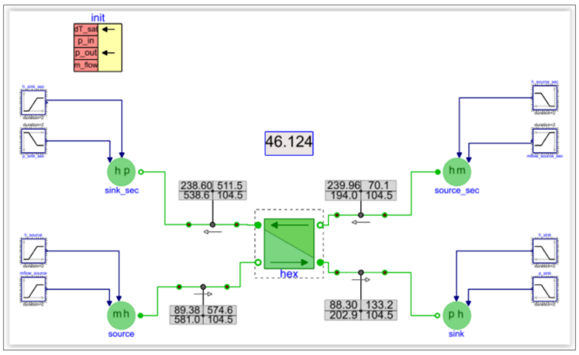


Figure 7. HTC test bench

2.3 Brayton Cycle Control

Sensor signals from relevant components are placed onto the control bus. A controller component is then connected to the primary heater and cooler heat sources. As shown in Figure 8 two PID controllers are used, one to provide heat to control turbine inlet temperature and other to control the temperature at the inlet of main compressor (to cool the fluid before entering compressor).

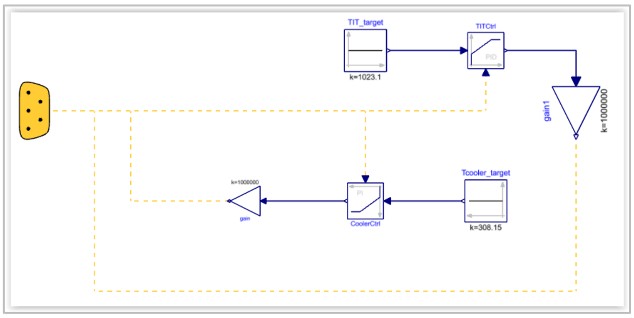


Figure 8. Brayton cycle controller

3 Simulation Results

Following the characterization of the system, a series of simulations were run by varying the turbine inlet temperature (TIT) in the range of 250 to 750 ^{0}C. Cycle efficiency variation with respect to change in turbine inlet temperature can be seen in Figure 9.

For turbine inlet temperature of 750 ^{0}C, power consumption of compressors and power generation by turbine are shown in Figure 10. Also heat transfer power of HTR, LTR, primary heater and cooler are shown in the same plot.

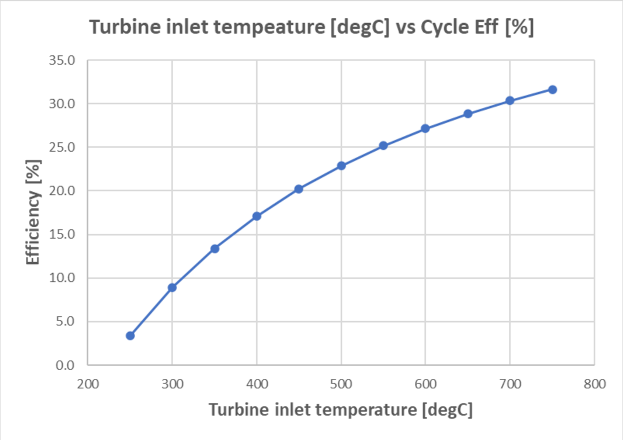


Figure 9. Turbine inlet temperature vs Cycle efficiency

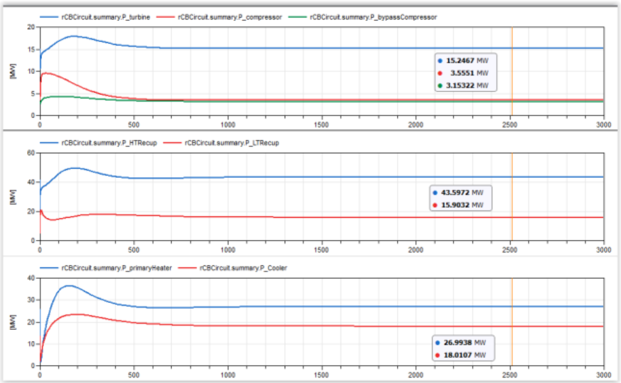


Figure 10. Different component powers

Pressure enthalpy diagram for the cycle at 750 ^{0}C turbine inlet temperature is shown in **Figure 11**. Pink line indicates the flow through bypass compressor and mixing with the main compressor outlet flow.

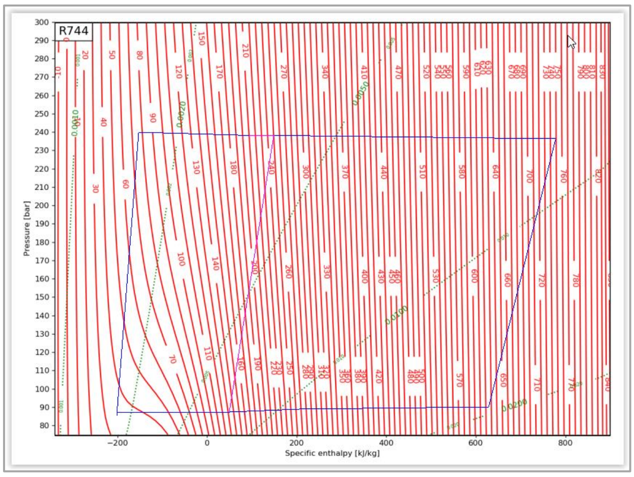
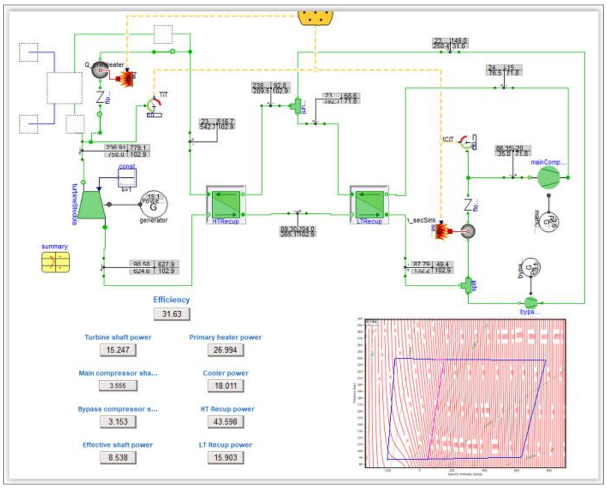


Figure 11. RCB PH diagram

Schematic of the cycle simulation with different visualizers is shown in Figure 12.



Efficiency
31.63

Turbine shaft power
15.247

Primary heater power
26.994

Main compressor sha...
3.555

Cooler power
18.011

Bypass compressor s...
3.153

HT Recup power
43.598

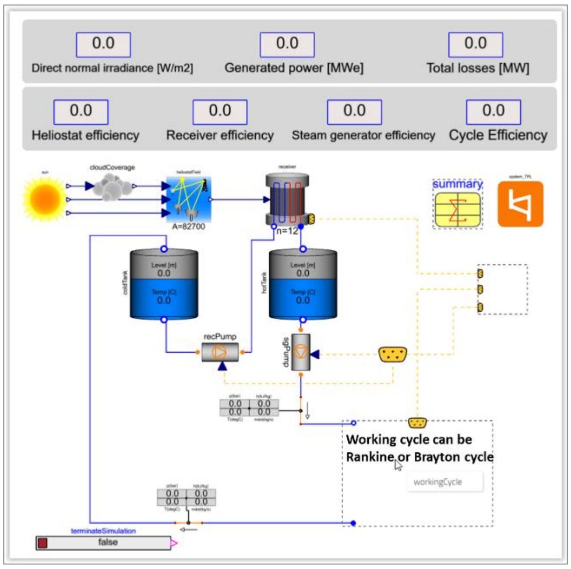
Effective shaft power
8.538

LT Recup power
15.903

Figure 12. Cycle schematic with results

4 CSP system

As mentioned in the introduction, CSP thermal storage system model with Rankine cycle is already available. This model is modified such that working cycle can be interchanged between Rankine and Brayton cycles as shown in Figure 13.



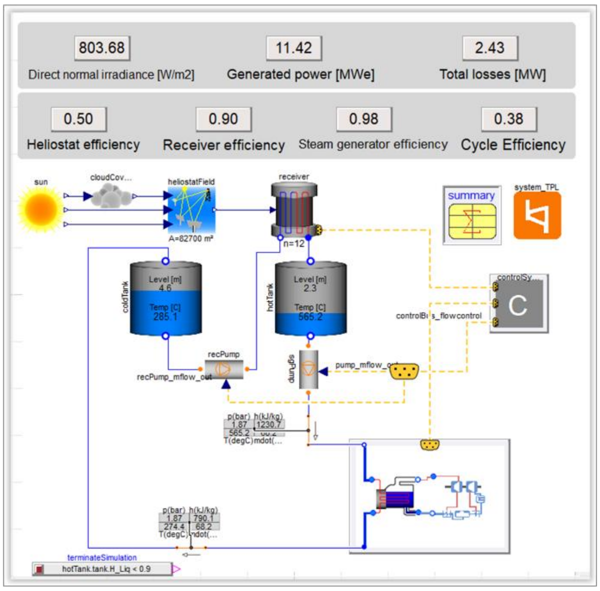
Direct normal irradiance [W/m2] 0.0
Generated power [MWe] 0.0
Total losses [MW] 0.0
Heliostat efficiency 0.0
Receiver efficiency 0.0
Steam generator efficiency 0.0
Cycle Efficiency 0.0

Level [m] 0.0
Temp [C] 0.0

Working cycle can be Rankine or Brayton cycle

Figure 13. CSP system with flexible working cycle

CSP thermal storage system with Rankine cycle and Brayton cycle are shown in Figure 14 and Figure 15 respectively. In the CSP system with Brayton cycle, heat exchanger is used as primary heater for interchanging heat between Molten salt and supercritical CO_2. As this option to change the routing through heat exchanger is already available, there is no requirement of modifying the original cycle.



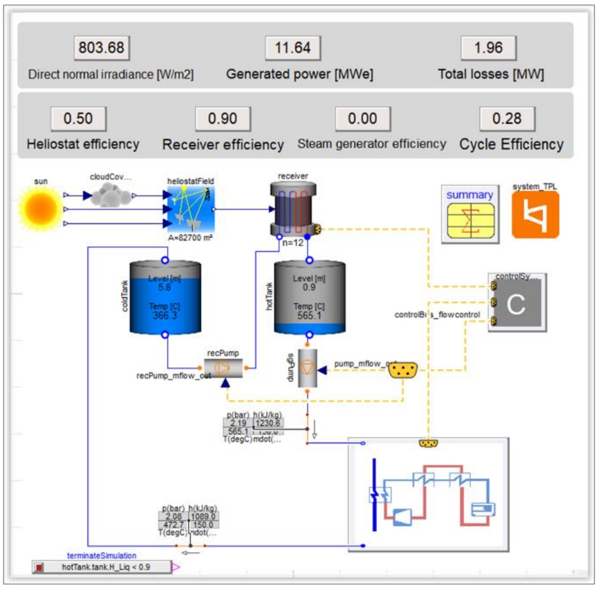
Direct normal irradiance [W/m2] 803.68
Generated power [MWe] 11.42
Total losses [MW] 2.43
Heliostat efficiency 0.50
Receiver efficiency 0.90
Steam generator efficiency 0.98
Cycle Efficiency 0.38

coldTank: Level [m] 4.6, Temp [C] 285.1
hotTank: Level [m] 2.3, Temp [C] 565.2

p(bar) 1.87, h(kJ/kg) 1230.7, T(degC) 565.2
p(bar) 274.4, h(kJ/kg) 68.2

hotTank.tank.H_Liq < 0.9

Figure 14. CSP system with Rankine cycle



803.68 Direct normal irradiance [W/m2]
11.64 Generated power [MWe]
1.96 Total losses [MW]
0.50 Heliostat efficiency
0.90 Receiver efficiency
0.00 Steam generator efficiency
0.28 Cycle Efficiency

Figure 15. CSP system with Brayton cycle

As can be seen from the visualizers, models are run under similar conditions on the solar plant side. And we can observe the cycle efficiencies for the Rankine and Brayton cycles. As the components are not characterized by the actual maps, the results cannot be used for comparison. But the idea is to show how different cycles can be used along with CSP system to evaluate the performance and efficiency of the system.

5 Summary

This paper describes an effort to develop an analytic platform for recompression Brayton cycle to evaluate the performance and efficiency under different operating conditions. This analytic platform allows rapid virtual prototyping to evaluate the potential of recompression Brayton cycle under different operating conditions and for different system sizing. As expected with increase in turbine inlet temperature, cycle efficiency is increasing. It is also shown how the Brayton cycle can be easily connected to CSP thermal storage system for evaluating the electric power generation through solar power. Also modular structure of Modelica language is taken advantage of to choose between connecting CSP thermal storage system to either Rankine or Brayton cycles. This helps in doing several studies side by side and for comparing the performances.

Future work on this model includes opportunities for better system characterization if data on the actual system can be obtained. In particular, actual characterization of the compressors, turbine and heat exchangers would greatly improve model accuracy. In this work results are shown for steady state simulations, future work includes dynamic response of the system under varying boundary conditions. Further study can also include comparison between the Ranking cycle and Brayton cycle in case actual system data is available for both the cycles.

References

Richard Dennis (2017). Overview of Supercritical Carbon Dioxide Based Power Cycles for Stationary Power Generation. International Seminar on Organic Rankine Cycle Power Systems; Politecnico di Milano; Milano Italy

Johan Edman, Johan Windahl (2015). Dynamic Modeling of a Central Receiver CSP system in Modelica. Proceedings of the 11th International Modelica Conference, pp. 586-594

Jim Pasch, Tom Conboy, Darryn Fleming, and Gary Rocahu (2012). Supercritical CO_2 Recompression Brayton Cycle: Complete Assembly Description. Sandia National Laboratories Report

Modelon AB, Lund, Sweden. (2020). *Vapor Cycle Library* Vapor Cycle Library | Modelica Library Built by Modelon

Modelon AB, Lund, Sweden. (2020). *Thermal Power Library* Thermal Power Library | Modelica Library Built by Modelon

Parallel Fast: An Efficient Coupling Approach for Co-Simulation with Different Coupling Step Sizes

Franz Holzinger[1,2] Klaus Schuch[2] Martin Benedikt[1] Daniel Watzenig[1,3]

[1]VIRTUAL VEHICLE Research GmbH, Austria,
{franz.holzinger,martin.benedikt,daniel.watzenig}@v2c2.at
[2]AVL List GmbH, Austria, {franz.holzinger,klaus.schuch}@avl.com
[3]Institute of Automation and Control, Graz University of Technology, Austria, daniel.watzenig@tugraz.at

Abstract

The primary task of co-simulation is the synchronization and exchange of data between the subsystems, e.g., FMUs, at certain coupling points. If the FMUs have different step sizes, the synchronization of the FMUs is often at the expense of the simulation duration of the co-simulation. The presented parallel fast scheduling algorithm is an effective approach to couple FMUs with different coupling step sizes. Therefore, synchronization intervals are introduced in which FMUs that finish their coupling step are synchronized. This allows a high performance of the coupling in terms of simulation duration. The higher performance compared to other scheduling algorithms is particularly evident in real-time applications, i.e., HiL simulations. However, the synchronization intervals are defined via a synchronization step size, which can be set independently to the coupling step sizes of the FMUs. This additional step size has a significant impact on the simulation accuracy. An extrapolation measure is introduced, which approximates the impact of the synchronization step size on the extrapolation error and thus on the simulation accuracy. Based on this, an optimization approach is presented, which derives the optimal synchronization step size to minimize the extrapolation measure.

parallel scheduling, synchronization step size, optimal step size

1 Introduction

In order to reduce development costs and development time, the focus in the industry has increasingly been placed on simulation in the recent decades. This led to a multitude of simulation environments to solve the engineering tasks in the different domains and applications. However, the different simulation tools often cover a specific area. In order to consider interactions across domains, it is necessary to integrate the specific models and tools into a combined simulation. In contrary to remodelling of the several specific models, which is cost or at least time intensive, co-simulation enables the direct integration of the individual subsystems and models, whereby coupling variables are exchanged at certain time steps to synchronize the subsystems as introduced in Kübler and Schiehlen (2000). Standardisations in the interface have reduced the technical effort to integrate subsystems as FMUs from different simulation environments, for more details see Blochwitz et al. (2012).

However, besides the technical implementation of the FMU[1] integration and data exchange, there are challenges in coupling and synchronization of the FMUs. Especially with non-iterative co-simulation, where coupling steps cannot be repeated, scheduling approaches, step-sizes and extrapolation techniques have a major impact on the simulation accuracy and the simulation duration.

In order to solve the causality problem between interacting FMUs extrapolation of coupling signals is needed. The most common representative is the ZOH (zero-order-hold) extrapolation, where the last known value of the coupling signals is used to determine the upcoming coupling step. The coupling imperfection resulting from the extrapolation can be handled with particular extrapolation and compensation techniques. For instance, a signal based extrapolation technique to compensate the energy losses caused by the coupling is discussed in Benedikt and Drenth (2019). In Haid et al. (2018) a model-based predictor corrector approach is introduced and a model-based pre-step stabilization technique is shown in Genser and Benedikt (2018).

In addition to the extrapolation techniques, the coupling step sizes, i.e., the defined time steps of the data exchange between the FMUs, contribute significantly to the simulation accuracy and the simulation duration. Evaluation and definition of the coupling step size based on the instantaneous frequency of the coupling signals is discussed in Benedikt, Watzenig, et al. (2013). Coupling error based adaptive step size approaches are analysed in Busch and Schweizer (2011) and Sadjina et al. (2016).

Beside them the scheduling influences the simulation accuracy and the simulation duration of the co-simulation. The scheduling can be categorized into two major groups: parallel and sequential coupling. Sequential coupling means that the FMUs are executed one after the other, which allows subsequent FMUs to access the results of

[1] For reasons of better understanding, the term FMU will be used for models and subsystems from now on. This is not a general restriction, all considerations are also valid for other model integrations.

DOI
10.3384/ecp21181649

FMUs that have already been executed. This reduces the extrapolations of the entire co-simulation and increases the simulation accuracy, but at the expense of the simulation duration. The simulation results strongly depend on the execution order of the FMUs. Approaches to identifying suitable execution sequences are discussed in Glumac and Kovacic (2018), F. Holzinger and Benedikt (2019), and Oakes et al. (2020).

With parallel coupling, the FMUs are executed simultaneously, which typically leads to shorter simulation durations than sequential co-simulation. However, the simulation accuracy typically decreases due to an increasing need of extrapolation. Nevertheless, for a performant simulation in terms of simulation time, e.g., for real-time applications, parallel coupling approaches are typically preferred.

However, not only the timing behaviour of the FMUs themselves defines the overall simulation duration. Especially if different coupling step sizes are used for the FMUs, dedicated scheduling algorithms are needed for the synchronization between the FMUs. These scheduling algorithms and their underlying synchronization strategy have an additional effect on the simulation duration and the simulation accuracy. For instance, Matlab Simulink uses a superior step size (solver step size) to synchronize the FMUs. This has the restriction, that the step sizes of the individual FMUs have to be a multiple of the superior step size, for more details see Matlab Simulink (2021). Especially with large differences in the step sizes of the FMUs, this can lead to unnecessarily increased synchronization efforts in terms of data exchange.

The presented parallel fast coupling approach also uses a superior step size, the so-called synchronization step size. However, the synchronization step size can be defined independently of the coupling step sizes of the FMUs, which enables a suitable data exchange between the FMUs. The parallel fast scheduling with the synchronization step size shows a performant simulation despite to different coupling step sizes. The selection of the synchronization step size has an effect on the data exchange and thus on the simulation accuracy. In order to achieve an suitable configuration, an optimization approach to define the synchronization step size based on an extrapolation assessment of the coupling signals is discussed.

This work is structured as follows: The next chapter introduces the parallel fast scheduling and discusses its execution behaviour. In the third chapter the extrapolation measure is introduced, which approximates the extrapolation error based on the topology and configuration of the co-simulation. Based on the extrapolation measure an optimization approach to identify the optimal synchronization step size for the parallel fast scheduling is derived in chapter four. A co-simulation example with four FMUs S_A, S_B, S_C, S_D is used to illustrate the properties of the parallel scheduling and the optimization approach. The topology of the co-simulation example is shown in Figure 1. The FMUs of the example are based on an example in Benedikt and Drenth (2019) and F. Holzinger and Benedikt (2019). All FMUs refer to FMU for co-simulation according to the FMI version 2.0. The parallel fast approach is available as a scheduling procedure in the AVL Co-Simulation platform Model.CONNECT™ for more information see Model.CONNECT™ (2020).

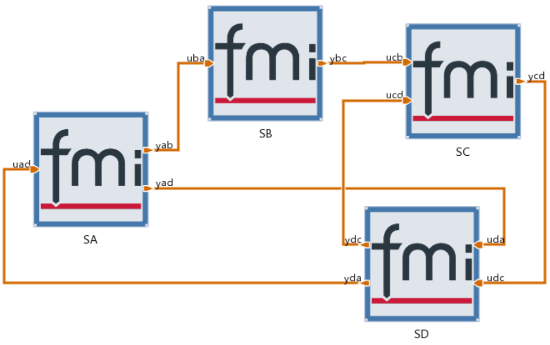


Figure 1. Co-Simulation topology of an example with four FMUs.

2 Parallel Fast Scheduling

Parallel scheduling algorithms execute the FMUs simultaneously. Due to the parallel execution, these approaches basically have a high performance in terms of simulation duration. Which makes them suitable for real-time applications, where the entire co-simulation must calculate faster than real-time. If the FMUs have the same coupling step sizes h_i, they have the identical simulation progress nh_i, i.e., simulation time $t_{s,i}$, after each coupling step n. With respect to the data exchange, the FMUs have to wait for each other after a coupling step. Neglecting the time for data exchange, the simulation duration and thus the real-time factor of the entire co-simulation is dominated by the slowest FMU and can be estimated as follows:

$$\hat{D} = \max\{d_i\}, \tag{1}$$

where d_i are the real-time factors of the individual FMUs S_i and $\hat{D}$ is the estimated real-time factor of the entire co-simulation. A real-time factor $\hat{D} = 1$ indicates that the co-simulation is running in real-time, i.e. the simulation time t_s is equal to the wall clock time t_w. If $\hat{D} < 1$, the co-simulation is faster than real time ($t_s < t_w$) and if $\hat{D} > 1$, the co-simulation is slower than real time ($t_s > t_w$). However, if different coupling step sizes of the FMUs are used, the simulation progress and thus the simulation time $t_{s,i}$ will differ for the individual FMUs S_i.

Depending on the coupling strategy the FMUs are synchronized at different time steps. Approaches, like the latest-first scheduling approach, where the several FMUs are synchronized after each coupling step, show a significant increasing of the simulation duration due to different coupling step sizes.

However, the presented parallel fast approach avoids such behaviour by introducing a global step size for synchronization, the so-called synchronization step size H.

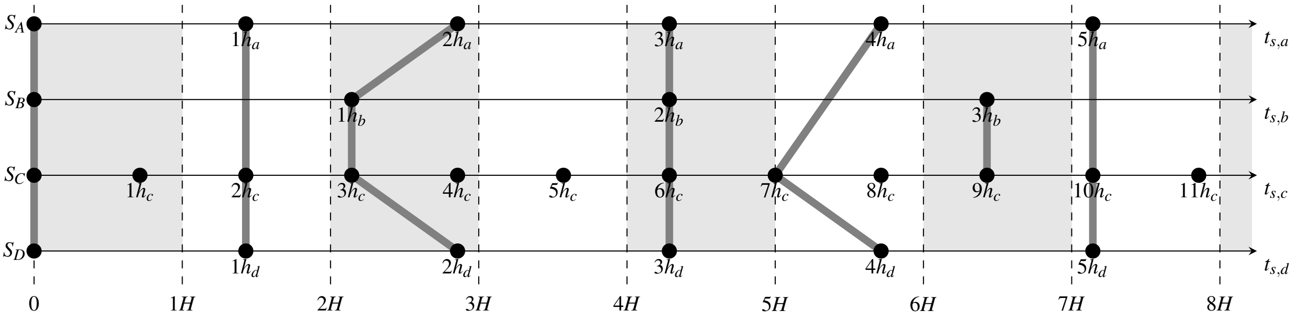


Figure 2. Execution sequence of the parallel fast scheduling.

The step size H defines synchronization intervals. If two or more FMUs end their coupling step within a synchronization interval $[\zeta H, (\zeta+1)H)$, data is exchanged between these systems. In general, this synchronizations condition for two FMUs S_i and S_j can be written as follows:

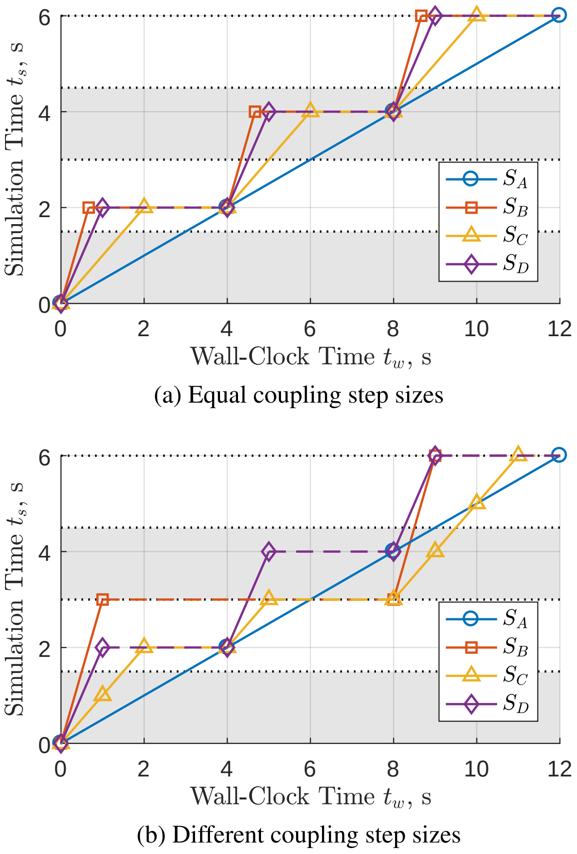
$$(n-1)h_i < \zeta H \leq nh_i < (\zeta+1)H \tag{2a}$$
$$(m-1)h_j < \zeta H \leq mh_j < (\zeta+1)H, \tag{2b}$$

where h_i and h_j represent the coupling step sizes of the FMUs S_i and S_j. A synchronization between two FMUs S_i and S_j takes place, if both FMUs end their coupling step nh_i and mh_j within the same synchronization interval $[\zeta H, (\zeta+1)H)$. In addition to that, it is mandatory, that the previous coupling step $(n-1)h_i$ and $(m-1)h_j$ is outside of the synchronization interval. This avoids multiple data exchange within a synchronization interval. However, the condition in (2) can lead to fewer synchronisations between the subsystems for small synchronisation step sizes, if the individual subsystems end their coupling steps at different synchronisation intervals.

Figure 2 shows the FMU execution and synchronization of a co-simulation example with four FMUs S_A, S_B, S_C and S_D. The FMUs have different coupling step sizes which are assumed with $h_a = 2s$, $h_b = 3s$, $h_c = 1s$ and $h_d = 2s$. The synchronization step size is set to $H = 1.4s$. The axes show the simulation progress of the individual FMUs with their coupling steps illustrated as cycles. The vertical dashed lines depict the synchronization step sizes. The synchronization and thus the data exchange between the FMUs is illustrated with dark grey solid lines. The first synchronization is in the synchronization interval $[H, 2H)$ at $t_s = 2s$. FMUs S_A, S_C and S_D end at the same time and exchange their data. The data exchange is done independently from the simulation progress of FMU S_B. The first synchronization (and data exchange between) all FMUs (including FMU S_B) occurs in the interval $[2H, 3H)$. In the following interval $[3H, 4H)$ only FMU S_C finishes its step. Since no other FMU ends the simulation step in this interval, there is no synchronization and FMU S_C continues the calculation without data exchange. This strategy will be continued for the further intervals. Depending on the coupling step of the FMU, synchronize all FMUs or only parts of the FMUs with each other.

2.1 Synchronization Interval

The synchronization step size H has a direct impact on the synchronization and data exchange of the individual FMUs and thus on their execution behaviour. Figure 3 shows the execution behaviour in terms of the simulation time t_s over the wall-clock time t_w of the co-simulation example in Figure 1 with the FMUs S_A, S_B, S_C and S_D. The solid lines show the execution of the FMUs with a marker at the beginning and the end of the coupling step. The



(a) Equal coupling step sizes

(b) Different coupling step sizes

Figure 3. Simulation time of the FMUs depending on the wall clock time with parallel fast scheduling: (a) Equal coupling step sizes; (b) Different coupling step sizes.

horizontal (dashed) lines between the markers indicate the waiting of the FMU due to the synchronization condition in 2. The synchronization step size is defined as $H = 1.5\,s$. The real-time factor of the FMUs is assumed with $d_a = 2$, $d_b = 0.333$, $d_c = 1$ and $d_d = 0.5$, i.e., for instance FMU S_A with a real-time factor $d_a = 2$ needs $4\,s$ to execute a coupling step of $h_a = 2\,s$. In Figure 3 a all FMUs have the same coupling step sizes $h_a = h_b = h_c = h_d = 2\,s$. Consequently, all FMUs are synchronized at the same time. The FMUs start their execution simultaneously. Due to the different real-time factors, the FMUs end their coupling step at different wall-clock times t_w. For instance, FMU S_C has finished the first coupling step at $t_w = 2\,s$, while FMU S_A ends the first step at $t_w = 4\,s$. After all FMUs have completed their coupling step, data is exchanged. Due to the equal step sizes and thus the equal simulation progress, all FMUs start their next coupling step again simultaneously.

The simulation time t_s over the wall-clock time t_w for different coupling step sizes is shown in Figure 3 b for assumed coupling step sizes $h_a = 2\,s$, $h_b = 3\,s$, $h_c = 1\,s$ and $h_d = 2\,s$. In the first coupling step all FMUs are executed in parallel. FMU S_C is the only FMU that finishes its first step in the synchronization interval $[0\,s, 1.5\,s)$. This means that no synchronisation is necessary and the FMU immediately executes the next coupling step. The first data exchange takes place between the FMUs S_A, S_C and S_D in the synchronization interval $[1.5\,s, 3\,s)$. The three FMUs wait for each other before starting the next coupling step after the data exchange. Synchronization with FMU S_B first occurs in interval $[3\,s, 4.5\,s)$. All FMUs complete their coupling step within this interval. Data is exchanged between all FMUs, despite the different simulation progresses. In the following interval $[4.5\,s, 6\,s)$, only FMU S_C fulfils the synchronization condition, which leads to no synchronization. As a result, FMU S_C executes three coupling steps in a row without exchanging data with the other FMUs. However, at $t_s = 6\,s$, i.e., within interval $[6\,s, 7.5\,s)$, all FMUs end their coupling step and synchronization is performed.

In both diagrams in Figure 3, the simulation duration or the wall-clock time $t_w = 12\,s$ for a simulation time $t_s = 6\,s$. That means that the slowest FMU, i.e., FMU S_A, dominates the simulation duration in both cases. Hence, the synchronization between the FMUs has no effect on the simulation duration in the shown example.

However, the synchronization step size H has an impact on the data exchange between the FMUs and thus on the simulation extrapolation behaviour. Figure 4 shows the extrapolation error E and the real-time factor D of the co-simulation example over the synchronization step size H. Thereby, the extrapolation error E and the real-time factor D of the entire co-simulation were determined from several simulation runs with different synchronization step sizes. The extrapolation error E results from the mean local extrapolation error of all inputs of the FMUs, i.e., it is a measure of the actual extrapolation errors of the entire co-simulation. To identify the impact of the synchronization step size H on the simulation duration and real-time factor D, it is assumed, that all FMUs have the identical real-time factor $d_a = d_b = d_c = d_d = 1$.

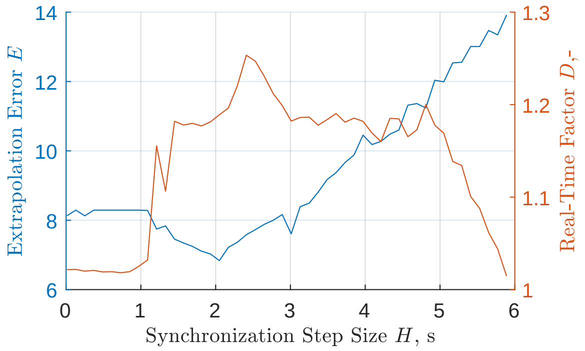


Figure 4. Real-time factor D and extrapolation error $\hat{E}$ of the actual co-simulation depending on the synchronization step size H of the parallel fast scheduling.

For small synchronization step sizes H, the real-time factor of the entire co-simulation is about $D = 1$. After that, the real-time factor increases to about $D = 1.2$, because of the larger synchronization interval, there may be waiting time between FMUs due to different simulation progress, which increases the simulation time of the whole co-simulation. With a synchronization step size of $H = 6\,s$, the real-time factor drops again to $D = 1$ in this example. The synchronization step size $H = 6\,s$ corresponds to the common multiple of all coupling step sizes, i.e., all FMUs end at the same simulation time t_s, which minimize the synchronization time between the FMUs.

The extrapolation error E is almost constant for small synchronization step sizes H in this example. From a synchronization step size of $H = 1\,s$, the extrapolation error E decreases and afterwards increases almost linearly from $H = 2\,s$. Larger synchronization intervals lead to less data exchange between FMUs. This typically means larger extrapolation errors and thus a decrease in simulation accuracy. The lowest extrapolation error and thus the highest accuracy is at $H = 2\,s$. Due to the synchronization condition in (2) smaller synchronization step sizes H do not necessarily lead to better results. For small step sizes, the coupling steps of the FMUs must be finished close to each other in order to be synchronized.

3 Coupling Assessment

The extrapolation error E and the real-time factor D in Figure 4 result from the execution behaviour of the parallel fast scheduling. In addition to the synchronization step size H this depends on the topology of the co-simulation, the coupling step sizes h_i of the individual FMUs and their timing behaviour d_i. Based on these parameters, an extrapolation measure $\hat{E}$ is introduced, which is called extrapolation error estimation. In addition to that the real-time factor $\hat{D}$ of the parallel fast scheduling for the entire co-simulation is estimated.

3.1 Extrapolation Assessment

The scheduling defines the synchronization depending on the connections between the FMUs, their coupling step step sizes and the synchronization step size. The synchronization results in the extrapolation behaviour of the coupling signals. In order to derive the extrapolation in the form of an extrapolation error estimation $\hat{E}$, synchronization of the FMUs must be considered. In this context, the interactions between the FMUs are essential. These interactions are defined by the coupling signals, i.e., the connection from an output y_i of FMU S_i to an input u_j of FMU S_j. For the entire co-simulation the connections can be described with the linking matrix:

$$\mathbf{u} = \mathbf{L} \cdot \mathbf{y}, \tag{3}$$

with the vector of all concatenated outputs $\mathbf{y} = [y_1, y_2, \ldots, y_N]^T$ and the vector of all concatenated inputs $\mathbf{u} = [u_1, u_2, \ldots, u_N]$. Without loss of generality, we assume that both vectors have the length N, where N represents the number of connections, i.e., an output can only be connected to one input[2]. Consequently, the linking matrix $\mathbf{L}$ is an orthogonal matrix with the dimension $N \times N$. The connections of the example in Figure 1 can be written using a linking matrix:

$$\begin{bmatrix} u_{ad} \\ u_{ba} \\ u_{cb} \\ u_{cd} \\ u_{dc} \\ u_{da} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} y_{ab} \\ y_{ad} \\ y_{bc} \\ y_{cd} \\ y_{da} \\ y_{dc} \end{bmatrix} \tag{4}$$

However, the linking matrix does not contain any information regarding the dependencies between the FMUs. In order to derive the dependency of the FMUs from the linking matrix, which describes the mapping of the inputs to the outputs, two further matrices $\mathbf{S}$ and $\mathbf{T}$ are introduced. The matrix $\mathbf{S}$ describes the relation of the outputs y_i with the corresponding FMUs S_i and the matrix $\mathbf{T}$ describes relation of the inputs u_j with the corresponding FMUs S_j. Based on the example, the matrices are as follows:

$$\mathbf{S} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{5}$$

The two matrices $\mathbf{S}$ and $\mathbf{T}$ have the dimension $N \times M$, where M is the number of involved FMUs. In combination with the linking matrix, the dependency matrix $\mathbf{A}$, i.e., the dependencies between the FMUs, is

$$\mathbf{A} = \left(\mathbf{T}^T \cdot \mathbf{L} \cdot \mathbf{S}\right)^T . \tag{6}$$

The dependency matrix $\mathbf{A}$ has the dimension $M \times M$. For the example in Figure 1, the dependency matrix can be written as follows:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}, \tag{7}$$

where the individual columns and rows are assigned to the FMUs S_A, S_B, S_C and S_D. The entries in the columns indicates the inputs to the FMUs and the rows the outputs. For instance, the first column, i.e., FMU S_A, represents the inputs of FMU S_A, which is connected to FMU S_D. The first row describes the two outputs of FMU S_A to the FMU S_B and S_D.

The dependency matrix $\mathbf{A}$ serves as the basis to determine the extrapolation error estimation $\hat{E}$. In a parallel coupling, where all FMUs S_i have the same coupling step sizes h_i, all inputs u_i are always extrapolated. With respect to the dependency matrix, the extrapolation error estimation can be stated as

$$\hat{E} = \sum_{i=1}^{M} \sum_{j=1}^{M} \mathbf{A}_{ij}. \tag{8}$$

With the definition of the dependency matrix in (6) the extrapolation error estimation $\hat{E}$ corresponds to the number of extrapolated inputs, which implies the assumption that all coupling signals have the same influence on the coupling error and thus on simulation results. Typically, there are coupling signals that have more influence on simulation results than others. This can be considered by extending the dependency matrix $\mathbf{A}$ in (6) with a weighting matrix $\mathbf{C}$ as follows:

$$\mathbf{A} = \left(\mathbf{T}^T \cdot \mathbf{C} \cdot \mathbf{L} \cdot \mathbf{S}\right)^T , \tag{9}$$

where the diagonal matrix $\mathbf{C} = diag(c_1, c_2, ..., c_n)$ has the dimension $N \times N$. The coefficient c_i represents the weighting of the coupling signals. A high weighting factor c_i shows a big impact of the coupling signal on the simulation results and a small weighting factor c_i indicates a little impact on the simulation results. Different methods to weight the coupling signals and their impact on the dependency matrix were discussed by the authors in F. Holzinger and Benedikt (2019) and F. R. Holzinger, Benedikt, and Watzenig (2021).

However, if different coupling step sizes are used, the extrapolation between FMUs and thus the extrapolation error E changes. Consequently, the extrapolation error estimation $\hat{E}$ in (8) is no longer valid, due to the fact that neither the coupling step sizes nor the scheduling is considered. The synchronization and thus the extrapolation

[2] In case of multiple connected outputs, the outputs can be duplicated.

of the FMUs depends on the simulation progress nh_i of the individual FMUs S_i according to the synchronization condition in (2).

In the case of a synchronization between two FMUs S_i and S_j, the synchronization step $\Delta\tau_{ij}^{(syn)}$ of FMU S_i is given by the actual simulation progress nh_i and previous simulation progress of the last synchronization $\tilde{n}h_i$:

$$\Delta\tau_{ij}^{(syn)} = nh_i - \tilde{n}h_i, \tag{10}$$

where $\tilde{n}$ indicates the last synchronization between the FMUs S_i and S_j. Due to the different coupling step sizes, synchronization between the FMUs does not take place at every coupling step nh_i. Furthermore, it is not mandatory that the FMUs have the same simulation progress during synchronization, they simply have to be in the same synchronization interval. The different simulation progresses during synchronization have an effect on the extrapolation. If a FMU is more progressed, its results can be used directly and do not have to be extrapolated by the other FMUs, i.e., the inputs can be interpolated. The interpolated part results from the simulation progress of the two coupled FMUs:

$$\Delta\tau_{ij}^{(int)} = mh_j - nh_i \quad \text{for} \quad mh_j > nh_i. \tag{11}$$

Considering FMU S_i, interpolation applies if the simulation progress mh_j of the coupled system S_j is greater than the simulation progress nh_i during the synchronization, otherwise the entire synchronization step for FMU S_i must be extrapolated. Therefore, the interpolated part can be generally written as follows:

$$\Delta\tau_{ij}^{(int)} = \max(0, mh_j - nh_i) \tag{12}$$

the extrapolated part $\Delta\tau_{ij}^{(ext)}$ of the synchronization results from the synchronization step in (10) minus the interpolated part in (12). The interpolated part cannot be larger than the synchronization step, which leads to the following:

$$\begin{aligned}\Delta\tau_{ij}^{(ext)} &= \Delta\tau_{ij}^{(syn)} - \Delta\tau_{ij}^{(int)} \\ &= nh_i - \tilde{n}h_i - \max(0, mh_j - nh_i)\end{aligned} \tag{13}$$

The extrapolated part $\Delta\tau_{ij}^{(ext)}$ in (13) is the time horizon, which is extrapolated within a synchronization step $\Delta\tau_{ij}^{(syn)}$. However, multiple coupling steps can be executed within one synchronization step and the number may vary from synchronization step to synchronization step. For instance, in the first synchronization step of FMU S_C in Figure 2, two coupling steps are executed, whereby in the next synchronization step only one coupling step is executed. The interpretation of the extrapolation error with the dependency matrix $\mathbf{A}$ in (9) is referred to a single coupling step. Due to lack of additional information about the coupling behaviour, it is assumed that the extrapolation impact increases linearly with the number of coupling steps within a synchronization step. That means, if there are multiple coupling steps within a synchronization step, the extrapolated part must be scaled by the impact of multiple coupling steps. For example, three coupling steps are executed within one synchronization step, i.e., $n - \tilde{n} = 3$. The first coupling step starts directly after the last synchronization, i.e., the input data has to be extrapolated only one coupling step size, so there is no scaling for the first coupling step needed. The second coupling step uses the same data for the extrapolation. The extrapolation horizon is two coupling steps in this case. Therefore, the second input value is scaled with the factor 2. In the third coupling step, the extrapolation horizon is already three coupling steps, i.e., scaling with a factor of three. The scaling factor f_i of the synchronization step can be written for the number of $n - \tilde{n}$ coupling steps as follows:

$$f_i = \frac{1}{n - \tilde{n}} \sum_{j=1}^{n-\tilde{n}_i} j = \frac{n - \tilde{n} + 1}{2} \tag{14}$$

However, the synchronization time between two FMUs and so the number of coupling steps can change from one synchronization step to the other. Therefore, the extrapolation must be reassessed for each synchronization. In order to derive an overall assessment of the extrapolation behaviour of the coupling signals between two FMUs, it is not sufficient to consider only one synchronization step. The synchronization behaviour and thus the extrapolation behaviour repeats after a certain time. This time corresponds to the least common step size $\bar{H}$ of the coupling step sizes h_i and h_j of the two FMUs and the synchronization step size H. For this time horizon $\bar{H}$, the mean extrapolation error estimation $\hat{E}$ can be calculated for a single coupling signal from FMU S_i to FMU S_j. This normalised extrapolation error estimation or extrapolation ratio $Q(h_i, h_j, H)$ describes the impact of the extrapolation of a coupling signal due to the synchronization step size H and the coupling step sizes h_i and h_j and can be written as:

$$Q(h_i, h_j, H) = \frac{1}{\bar{H}} \sum_{\zeta=1}^{\bar{H}/H} \sum_{n=1}^{\bar{H}/h_i} \sum_{m=1}^{\bar{H}/h_j} s_{e,i} s_{e,j} f_i \Delta\tau_{ij}^{(exp)}, \tag{15}$$

where f_i is the scaling factor regarding the synchronization step in (14), $s_{e,i}$ is the synchronization coefficient for FMU S_i and $s_{e,j}$ is the synchronization coefficient for FMU S_j given as:

$$s_{e,i} = \begin{cases} 1 & \text{if} \quad (n-1)h_i < \zeta H \leq nh_i < (\zeta+1)H \\ 0 & \text{otherwise and} \end{cases} \tag{16a}$$

$$s_{e,j} = \begin{cases} 1 & \text{if} \quad (m-1)h_j < \zeta H \leq mh_j < (\zeta+1)H \\ 0 & \text{otherwise.} \end{cases} \tag{16b}$$

The coefficient $s_{e,i} = 1$ if the condition in (2a) for FMU S_i is fulfilled. Analogously, the coefficient $s_{e,j} = 1$ if the synchronization condition in (2b) for FMU S_j is fulfilled.

The extrapolation ratio $Q(h_i, h_j, H)$ describes the effect of the parallel fast scheduling on the dependency matrix $\mathbf{A}$ and thus on the extrapolation error estimation $\hat{E}$. The weights of the extrapolation ratio $Q(h_i, h_j, H)$ for the individual step sizes $[h_1, h_2, ..h_M]$ of the individual FMUs can be put together in weighting matrix as follows:

$$\mathbf{W} = \begin{bmatrix} Q(h_1,h_1,H) & \cdots & Q(h_1,h_M,H) \\ \vdots & \ddots & \vdots \\ Q(h_M,h_1,H) & \cdots & Q(h_M,h_M,H) \end{bmatrix} \tag{17}$$

The weighting matrix $\mathbf{W}$ can be element-wise multiplied with the dependency matrix $\mathbf{A}$ to scale it according to the influence of the synchronization of the parallel fast algorithm. Whereby the extrapolation error estimation $\hat{E}$, i.e., the assessment of the extrapolation to the co-simulation, can be determined as follows:

$$\hat{E} = \sum_{i=1}^{M}\sum_{j=1}^{M} \bar{\mathbf{A}}_{ij} = \sum_{i=1}^{M}\sum_{j=1}^{M} \mathbf{W}_{ij}\mathbf{A}_{ij}. \tag{18}$$

3.2 Simulation Duration Assessment

The time of synchronization is defined by the synchronization step size H and the coupling step sizes h_i of the FMUs S_i. The actual duration (relative to wall-clock time) of the synchronization step depends on the duration of the execution of a coupling step nh_i, i.e., of the real-time factor d_i of the FMUs S_i, and the time the FMUs have to wait for each other during synchronization. Therefore, a coupling step implies mainly two parts: the execution of the FMU, dominated by the real-time factor d_i of the FMUs S_i and the synchronization between the FMUs, i.e., the waiting time. During this time, the synchronizing FMUs wait until all have completed their coupling step. The wall-clock time $t_{w,i}$ of the individual FMUs, i.e., the time that is actually needed to execute a coupling step, can be formulated depending on their simulation progress t_s as follows:

$$t_{w,i}(t_s) = \max\left(t_{w,i}(t_s), t_{w,j}(t_s)s_{e,i}s_{e,j}\right) \qquad \forall i,j \in S \tag{19}$$

The actual wall-clock time $t_{w,i}$ of a FMU S_i is determined as the maximum wall-clock time $t_{w,j}$ of all synchronized FMUs S_j. The synchronized FMUs are specified with the coefficients $s_{e,i}$ and $s_{e,j}$ regarding to the synchronization condition in (16). The simulation time can be expressed as $t_s = k\bar{h}$, where $k \in \mathbb{N}$ and $\bar{h}$ is the great-common-divisor step size of the coupling step sizes h_i and the synchronization step size H and consequently the determination of the wall-clock-time in (19) can be rewritten as follows:

$$t_{w,i}(k\bar{h}) = \max\left(t_{w,i}(k\bar{h}), t_{w,j}(k\bar{h})s_{e,i}s_{e,j}\right) \qquad \forall i,j \in S. \tag{20}$$

Depending on the definition of the simulation time $t_s = k\bar{h}$, the index n of the actual coupling step of a FMU S_i can be determined as $n = \left\lceil \frac{t_s}{h_i} \right\rceil = \left\lceil \frac{k\bar{h}}{h_i} \right\rceil$. Similarly, the index of the synchronization step is $\zeta = \left\lfloor \frac{t_s}{H} \right\rfloor = \left\lfloor \frac{k\bar{h}}{H} \right\rfloor$. Applied to (2a) the synchronization condition results in

$$\left(\left\lceil \frac{k\bar{h}}{h_i} \right\rceil - 1\right) h_i < \left\lfloor \frac{k\bar{h}}{H} \right\rfloor H \leq \left\lceil \frac{k\bar{h}}{h_i} \right\rceil h_i < \left(\left\lfloor \frac{k\bar{h}}{H} \right\rfloor + 1\right) H.$$

The wall-clock time $t_{w,i}$ of a FMUs S_i is individually updated with each execution by its real-time factor d_i. This can be generally written as follows:

$$t_{w,i}((k+1)\bar{h}) = t_{w,i}(k\bar{h}) + d_i h_i s_{d,i} \qquad \forall i \in S, \tag{21}$$

where $s_{d,i}$ indicates the end of a coupling step and can be stated as:

$$s_{d,i} = \begin{cases} 1 & \text{if} \quad \bmod(k\bar{h}, h_i) = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

If a coupling step is completed, i.e., the simulation time t_s or $k\bar{h}$ is a multiple of the coupling step size h_i, the coefficient $s_{d,i} = 1$ and its required computation effort $d_i h_i$ is added to the current wall-clock time $t_{w,i}$ of FMU S_i.

In order to approximate the real-time factor of the entire co-simulation it is not sufficient to consider only one synchronization step. As shown in Figure 2, the synchronization between FMUs and so the computational effort changes depending on the simulation progress. However, the synchronization pattern and so the timing repeats after a certain time. This time depends on the coupling step sizes h_i and the synchronization step size H and can be determined by the least-common-multiple step size $\bar{H}$ of all step sizes. That means, to approximate the real-time factor $\hat{D}$ of the entire co-simulation, at least the simulation time interval $t_s = [0, \bar{H}]$ has to be considered.

Finally the estimated real-time factor $\hat{D}$ for the parallel fast scheduling can be determined as

$$\hat{D} = \frac{1}{\bar{H}} \max_{i \in S}(t_{w,i}). \tag{23}$$

Using the extrapolation error estimation $\hat{E}$ in (18) and the real-time factor estimation $\hat{D}$ in (23), the impact of the synchronization step size H on the extrapolation error and the simulation duration can be estimated without costly simulations. The comparison of the extrapolation error estimation $\hat{E}$ and the real-time factor estimation $\hat{D}$ with the actual extrapolation error E and the real-time factor D depending on the synchronization step size H is shown in Figure 5. Both, the extrapolation error estimation $\hat{E}$ and the estimation of the real time factor $\hat{D}$ of the entire co-simulation, show a similar behaviour than the actual extrapolation error E and the actual real-time factor D depending on the synchronization step size H.

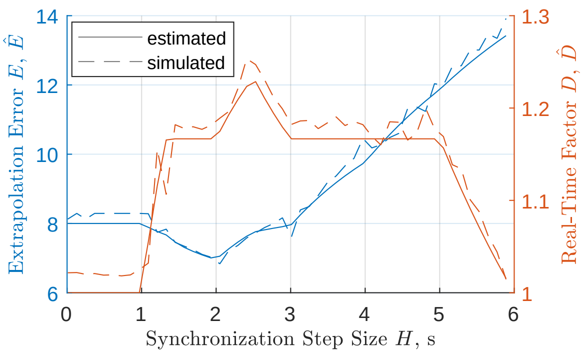


Figure 5. Real-time factor estimation $\hat{D}$ and extrapolation error estimation $\hat{E}$ compared to the actual real-time factor D and extrapolation error E depending on the synchronization step size H of the parallel fast scheduling.

4 Optimal Synchronization Step Size

The extrapolation error E changes strongly depending on the synchronization step size H, see Figure 5. Beyond a certain synchronization step size H, the extrapolation error increases almost linearly with the synchronization step size. This can be mainly explained by the large synchronization intervals that lead to less synchronizations, i.e., less data exchanges between the FMUs, and thus to higher coupling errors. However, the real-time factor D of the entire co-simulation changes only slightly with the synchronization step size H. The effect on the real-time factor and thus on the simulation duration becomes even smaller as soon as the FMUs have different real-time factors d_i. Figure 6 shows the impact of the synchronization step size H on the estimated real-time factor $\hat{D}$ for different real-time factors d_a of FMU S_A, whereby the other FMUs remain with a real-time factor $d_b = d_c = d_d = 1$.

With an increasing real-time factor d_a, the impact of the synchronization step size on the overall real-time factor $\hat{D}$ decreases. At a difference of around 60% of the real-time factor, i.e., $d_a = 1.6$, there is no significant impact of the synchronization time on the overall real-time factor D of the co-simulation. The slowest FMU, i.e., S_A, dominates the entire timing behaviour, independent on the synchronization step size H.

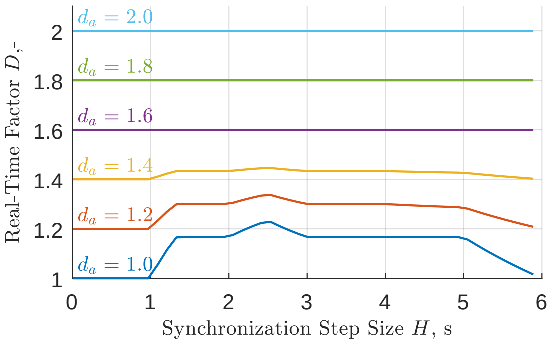
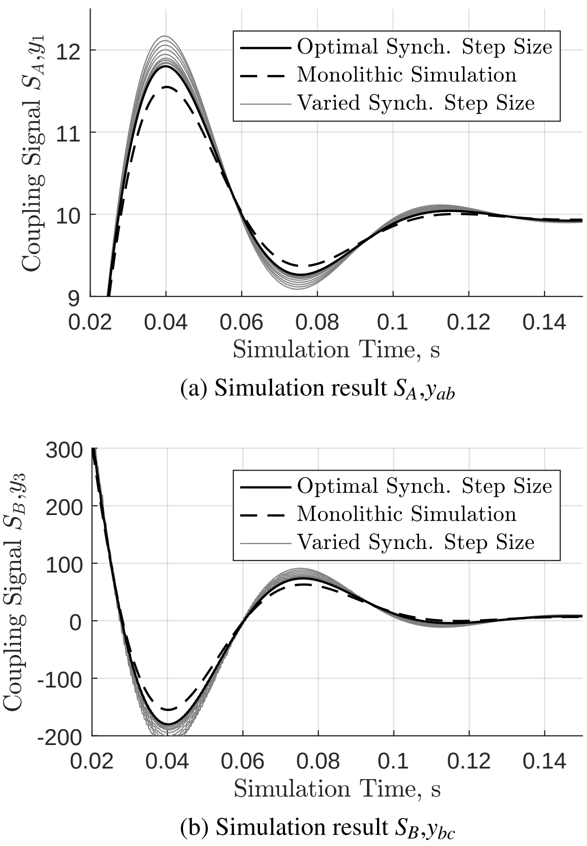


Figure 6. Real-time factor estimation depending on the synchronization step size for different real-time factors of FMU S_A.

Due to the small effect of the synchronization step size on the timing behaviour and thus on the overall real-time factor $\hat{D}$, the simulation duration is negligible for the determination of the synchronization step size. This means only the extrapolation error estimation $\hat{E}$ is used to find an appropriate synchronization step size. Thus, an optimization problem based on the extrapolation error estimation $\hat{E}$ in (18) can be written as follows:

$$\min_{H} \{\hat{E}\} . \tag{24}$$

A synchronisation step size H is desired that minimizes the extrapolation error estimation $\hat{E}$ in (18) for the considered co-simulation example. In the following, the optimization approach in (24) is applied to the co-simulation example in Figure 1. The coupling step sizes of the FMUs are given with $h_a = 0.2\,ms$, $h_b = 0.3\,ms$, $h_c = 0.1\,ms$ and $h_d = 0.2\,ms$. A detailed description of the FMUs can be found in Benedikt and Drenth (2019) and F. Holzinger and Benedikt (2019). To identify the optimal synchronization step size H, the minimum of the extrapolation error estimation has to be determined. Therefore 20 synchronization steps sizes were evaluated between $H = 0.05\,ms$ and



(a) Simulation result S_A,y_{ab}

(b) Simulation result S_B,y_{bc}

Figure 7. Simulation results for varied synchronization step sizes H, monolithic simulation and optimal synchronization step size H^*: (a) S_A,y_{ab}; (b) S_B,y_{bc}.

$H = 6\,ms$. The optimal synchronization step size based on the extrapolation error estimation $\hat{E}$ is $H^* = 2\,ms$.

Figure 7 shows the simulation results of the coupling signals y_{ab} and y_{bc} depending on the different synchronization step sizes. The black solid line depicts the simulation result of the optimal synchronization step size H^*. The grey lines show the results of the other evaluated synchronization step sizes. The monolithic simulation, i.e., all subsystems are solved within one model with one single solver without extrapolation, is shown as black dashed line and serves as reference signal. The results with the optimal synchronization step size H^* show the smallest deviation from the reference solution for the coupling signal y_{ab} of FMU S_A and coupling signal y_{bc} for FMU S_B.

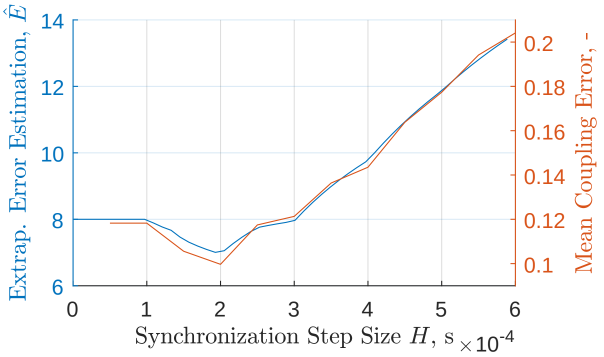


Figure 8. Extrapolation error estimation $\hat{E}$ compared to the mean coupling error w.r.t. the monolithic simulation over synchronization step size H.

The comparison of the determined extrapolation error estimation $\hat{E}$ with the mean coupling error is shown in Figure 8. The mean coupling error corresponds the root mean square error of the simulation results to the reference solution (monolithic simulation) of all coupling signals. The behaviour of the extrapolation error estimation $\hat{E}$ shows similar behaviour to the actual coupling error with respect to the monolithic simulation that occurs. This means that a determination of the optimal synchronization step size H^* using the extrapolation error estimation $\hat{E}$ leads to the optimal solution in terms of the synchronization step size.

5 Conclusion

The co-simulation of FMUs with different coupling step sizes requires suitable synchronization methods to ensure appropriate data exchange of the coupling signals between the FMUs. The presented parallel fast scheduling shows an effective way to synchronize the individual FMUs without unnecessarily extending the simulation duration. The usage of synchronization intervals, where FMUs exchange their data, enables continuous timing behaviour throughout the simulation and is therefore particularly suitable for real-time applications. However, the definition of the synchronization interval has a direct impact on the extrapolation behaviour of the FMUs and thus on the simulation accuracy. Contrary to the expectations, small synchronization intervals do not generally lead to accurate simulation results. Therefore, an optimisation approach was proposed to determine the optimal step size for the synchronization intervals based on an introduced extrapolation measure (extrapolation error estimation). This allows to derive the optimal synchronization step size for the parallel fast scheduling.

The presented determination of the optimal synchronization step size for the parallel fast scheduling based on the estimated extrapolation error shows good results for the example used in this work. However, in future work this approach will be analysed and validated on further examples. The extrapolation assessment depending on the synchronization step size of the co-simulation assumes a ZOH extrapolation for the estimation. The estimation of the extrapolation error for other coupling approaches, e.g., FOH, will be considered in future works.

Acknowledgements

The publication was partly written in cooperation with the VIRTUAL VEHICLE Research GmbH in Graz and partly funded by the COMET K2 – Competence Centers for Excellent Technologies Programme of the Federal Ministry for Transport, Innovation and Technology (bmvit), the Federal Ministry for Digital and Economic Affairs (bmdw), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG).

The work presented in this document is co-funded by the EU research and innovation programme Horizon2020 (VISION-xEV: GA No. 824314). The content does not reflect the official opinion of the European Union. Responsibility for the information and views expressed in the report lies entirely with the author(s).

References

Benedikt, Martin and Edo Drenth (2019). “Relaxing Stiff System Integration by Smoothing Techniques for Non-iterative Co-simulation”. In: *IUTAM Symposium on Solver-Coupling and Co-Simulation*. Ed. by Bernhard Schweizer. Cham: Springer International Publishing, pp. 1–25. ISBN: 978-3-030-14883-6.

Benedikt, Martin, Daniel Watzenig, et al. (2013). “Macro-step-size selection and monitoring of the coupling errof for weak coupled subsystems in the frequency-domain”. In: *Proceedings of International Conference on Computational Methods for Coupled Problems in Science and Engineering*, pp. 1–12. ISBN: 978-84-941407-6-1.

Blochwitz, T. et al. (2012). “Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models”. In: *Proceedings of the 9th International Modelica Conference*. Munich, Germany: The Modelica Association, pp. 173–184. ISBN: 978-91-7519-826-2. URL: http://dx.doi.org/10.3384/ecp12076173.

Busch, Martin and Bernhard Schweizer (2011). “An explicit approach for controlling the macro-step size of co-simulation methods”. In: *Proceedings of the 7th European Nonlinear Dynamics Conference (ENOC 2011): July 24 - 29, 2011,* *Rome, Italy*. Ed. by D. Bernadini, pp. 1–6. ISBN: 978-88-906234-2-4. URL: http://tubiblio.ulb.tu-darmstadt.de/77923/.

Genser, Simon and Martin Benedikt (2018). “A Pre-Step Stabilization Method for Non-Iterative Co-Simulation and Effects of Interface-Jacobians Identification”. In: *Advances in Intelligent Systems and Computing*. ISSN: 2194-5357.

Glumac, Slaven and Zdenko Kovacic (2018). “Calling Sequence Calculation for Sequential Co-simulation Master”. In: *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. SIGSIM-PADS ’18. Rome, Italy: ACM, pp. 157–160. ISBN: 978-1-4503-5092-1. DOI: 10.1145/3200921.3200924. URL: http://doi.acm.org/10.1145/3200921.3200924.

Haid, Timo et al. (2018). “A model-based corrector approach for explicit co-simulation using subspace identification”. In: The 5th Joint International Conference on Multibody System Dynamics.

Holzinger, Franz and Martin Benedikt (2019). “Optimal Trigger Sequence for Non-Iterative Co-Simulation:” in: *Proceedings of the 9th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. 9th International Conference on Simulation and Modeling Methodologies, Technologies and Applications. Prague, Czech Republic: SCITEPRESS - Science and Technology Publications, pp. 80–87. ISBN: 978-989-758-381-0. DOI: 10.5220/0007833800800087.

Holzinger, Franz Rudolf, Martin Benedikt, and Daniel Watzenig (2021). “Optimal Trigger Sequence for Non-iterative Co-simulation with Different Coupling Step Sizes”. In: *Simulation and Modeling Methodologies, Technologies and Applications*. Ed. by Mohammad S. Obaidat, Tuncer Ören, and Helena Szczerbicka. Cham: Springer International Publishing, pp. 83–103. ISBN: 978-3-030-55867-3.

Kübler, R. and W Schiehlen (2000). “Modular Simulation in Multibody System Dynamics”. In: *Multibody System Dynamics* 4.

Matlab Simulink, R2021a (2021). *Co-Simulation Execution and Numerical Compensation*. (https://www.mathworks.com/help/simulink/ug/co-simulation-execution-and-numerical-compensation.html). Accessed: 2021-04-11.

Model.CONNECT™, R2020a (2020). *AVL’s open model integration and co-simulation platform*. (https://www.avl.com/-/model-connect). Accessed: 2020-02-04.

Oakes, Bentley et al. (2020). “Hint-Based Configuration of Co-simulations with Algebraic Loops”. In: *Simulation and Modeling Methodologies, Technologies and Applications*. Springer International Publishing, pp. 1–28. ISBN: 978-3-030-55866-6. DOI: 10.1007/978-3-030-55867-3_1.

Sadjina, Severin et al. (2016-11). “Energy conservation and power bonds in co-simulations: non-iterative adaptive step size control and error estimation”. In: *Engineering with Computers* 33.3, pp. 607–620. ISSN: 1435-5663. DOI: 10.1007/s00366-016-0492-8. URL: http://dx.doi.org/10.1007/s00366-016-0492-8.

Towards an automated generator of urban building energy loads from 3D building models

Alessandro Maccarini[1] Michael Mans[2] Christian G. Sørensen[1] Alireza Afshari[1]

[1]Department of the Built Environment, Aalborg University, Denmark, amac@build.aau.dk
[2]Institute for Energy Efficient Buildings and Indoor Climate, RWTH Aachen University, Germany

Abstract

Buildings in cities are one of the major contributors of carbon emissions worldwide. Thus, improving building energy efficiency is one of the key strategies towards sustainable urbanization. Urban building energy modeling (UBEM) is a valuable methodology to tackle these challenges, as it provides users with the energy demand of the building stock, scenarios evaluation, peak loads and other useful analyses. This paper presents an open-source tool to automatically convert 3D building models into ready-to-run Modelica models for urban energy simulations. The software enables users to create 3D building geometries, perform data enrichment and execute model generation of reduced order Modelica models. The software is written in Python and it has been developed as an add-on for the 3D creation application Blender. The first part of the paper describes the general approach and the architecture of the tool. In the second part, a demonstration of the tool's capabilities is illustrated.

Keywords: urban energy modeling, workflow automation, 3D visual editing, Modelica code generation

1 Introduction

Building energy modeling (BEM) tools are a key asset in the design of energy efficient buildings. Based on a mathematical model that describes the interaction between a building and its energy system, these tools perform simulations and calculate outputs in terms of energy use and thermal comfort (Hensen and Lamberts 2019).

Recently, the open-source, object-oriented and equation-based modeling language Modelica has become increasingly used in the field of BEM. The use of Modelica has not only grown in the field of BEM, but also in the field of urban building energy modeling (UBEM), where the focus is on the analysis of building stocks and district energy systems. One of the reasons of the increased used of Modelica in BEM and UBEM is that future building and district energy systems integrate multi-domain interconnected subsystems (thermal, hydraulic, electric and control) based on renewable energy generation, for which Modelica provides an appropriate single platform for modeling and simulation. Another reason is that Modelica simulation environments provide a 2D graphical modeling approach that fits well the design of buildings and district energy system topologies. On the other hand, such a 2D graphical approach is inconvenient for modeling the 3D shapes of buildings.

When working on the modeling of a building, Modelica users typically need to abstract the 3D geometry of a building thermal zone into a string of code, which contains area and orientation of the building surfaces. This manual configuration of a thermal building zone is an error-prone process and does not line up with the design workflow of architects and engineers, who are using BIM-based CAD tools such as ArchiCAD or Revit for their 3D building designs. For these reasons, in the last years, different research activities have been focused on the automatic generation of Modelica building energy models from International Foundation Class (IFC) files, which are output of BIM softwares.

Thorade et al. (2015) proposed a toolchain using the commercial simulation tool Simergy. In the first step, the user imports the IFC file as input, adds relevant data with the Simergy graphical user interface (GUI), and then exports the data set as SimModel, which is a data domain model which is able to store information for building energy simulation. In the next step, a mapping tool takes the SimModel file and a Python script generates the Modelica model through a template approach. Reynders et al. (2017) described a tool based on a Python framework. It can read IFC-files, determine the building topology for multi-zone building models, and generate Modelica building models for the Modelica IDEAS library. Nytsch-Geusen et al. (2019) developed an open-source toolchain which can transfer BIM models of 3D building constructions into executable thermal multi-zone Modelica buildings energy models. For this purpose, different open-source libraries and tools were integrated into a Python-based software architecture of the toolchain.

In the above mentioned studies, the focus was on the automatic generation of Modelica models for single buildings energy analyses. In terms of urban energy analyses, Remmen et al. (2018) developed TEASER, a Python-based automated framework that includes data enrichment, data processing and Modelica model generation for urban context. The tool can produce ready-to-run Modelica building models based on a low order thermal zone model using the IBPSA core library and their larger user libraries Buildings (Wetter et al. 2014), BuildingSystems

DOI
10.3384/ecp21181659

(Nytsch-Geusen et al. 2012), IDEAS (Jorissen et al. 2018) and AixLib (Müller et al. 2016). TEASER can be used with an archetype building approach, which needs only basic input data about a building, but can also use detailed building information for the modeling process. Nevertheless there is no GUI provided for the user, not for parameter settings (e.g. material properties, ventilation rates, etc.) nor for 3D building modeling.

To address the challenges posed by manual translation of 3D building models to Modelica models at urban scale, this paper presents a "Blender-based Automatic Generator of Energy Loads" (BAGEL). BAGEL is an open-source Python-based tool that uses the 3D creation platform Blender as a host application to provide 3D modeling, intuitive GUI and parametrization capabilities for automatic generation of Modelica-based building energy models. The target audience of the tool are scientists and engineers who aim at predicting heating and cooling energy loads of new and existing urban neighborhoods and communities. The paper begins with a description of BAGEL architecture and functionalities, and then proceeds by giving a demonstration of the tool's capabilities with a use case.

2 Methodology

This chapter introduces Blender as a host application and then presents the architecture of BAGEL for the conversion of 3D building geometries into Modelica models for urban energy simulations.

2.1 Blender

Blender (Blender Online Community 2021) is a free and open-source 3D creation platform. Its functionality includes mesh-based 3D modeling, advanced materials and texture specification, physically based rendering and a Python Application Programming Interface (API) amongst other features. The Python API is deeply integrated allowing, for example, specification of data, control over mesh elements and manipulation of the Blender interface. This enables the development of third-party software (called add-ons in Blender) which function inside Blender and expand its functionality. Some of these features have made Blender an increasingly popular host application for a range of scientific visualisation and analysis tools (Pyka et al. 2010; Scianna 2013; Kent 2013). In the field of building energy simulation, the VI-Suite add-on supports the conversion of geometry and construction materials to the EnergyPlus input format (Southall and Biljecki 2017).

2.2 BAGEL architecture

The architecture of BAGEL is shown in Figure 1. Once installed, BAGEL appears as a visual panel directly accessible in the 3D environment of Blender. It consists of three modules:

1. 3D building shaping
2. Data enrichment
3. Modelica model generation

2.2.1 3D building shaping

In Blender, the geometry of a scene is constructed from one or more objects. These objects can range from basic 3D shapes and lights to illuminate the scene to cameras to take pictures or make video. The relevant object type in the context of BAGEL is the so-called `mesh`. A `mesh` is a collection of vertices, edges and faces that describe a 3D shape. The BAGEL module *3D building shaping* allows users to automatically create a new building, which is represented by a 3D `mesh` whose geometrical and spatial properties can be freely modified using the dedicated visual commands. Such properties are:

- Dimensions along the three axes
- Rotation around the z-axis (i.e. orientation)
- Location in the scene

BAGEL is currently able to handle only simple building geometries with a low Level of Detail (LoD). The LoD concept is defined in the CityGML standard and defines building models at different levels of complexity and granularity of the geometric representation (Open Geospatial Consortium 2012). In particular, in BAGEL, buildings can only be represented by rectangular prisms, which correspond to LoD1.

2.2.2 Data enrichment

At this point, buildings are described in terms of geometrical and spatial design. However, to perform dynamic energy simulations it is necessary to enrich the models with additional properties such as building mass and thermal characteristics of the building envelope. The BAGEL module *Data enrichment* enables users to define a set of nine properties for each building. This set of properties resembles the input parameters that are needed to solve the resistance-capacitance model described in the ISO 13790 standard (International Standard Organization 2008). This model describes the thermo-physical behavior of buildings by means of an equivalent electric circuit consisting of five resistances and one capacity (5R1C). Such a model has been developed in Modelica language and it will be used as target Modelica model in the next step of the process, where the enriched Blender model will be translated into Modelica code. More details about the 5R1C Modelica model are provided in section 2.2.3

The parameters required to solve the 5R1C model can be assigned visually to each building in the 3D Blender scene through the BAGEL interface, which stores such parameters within the data structure of the respective building as `custom properties`. In Blender, `custom properties` are a way to store metadata in data-blocks

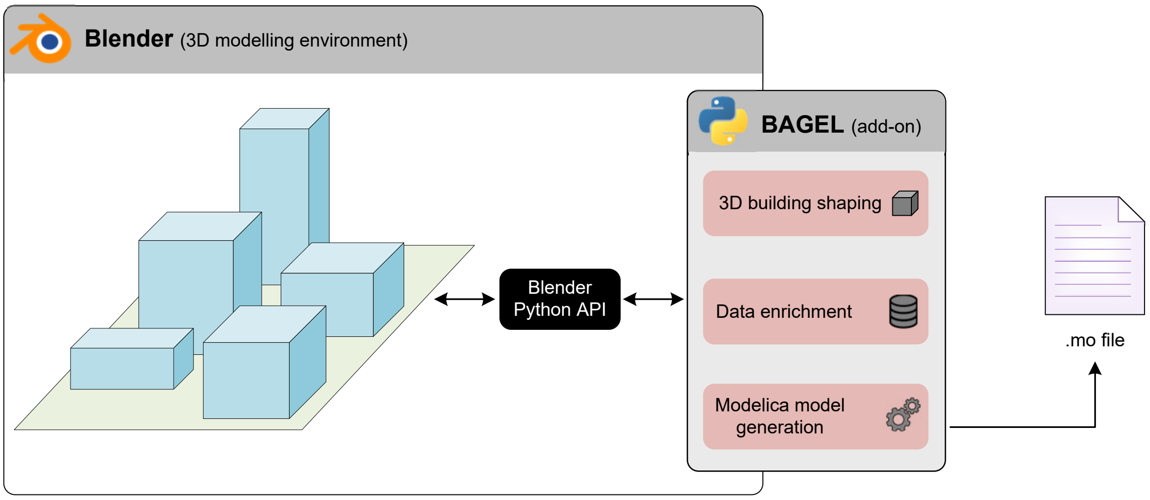


Figure 1. Software architecture of BAGEL.

(e.g. `mesh`) which can be used by Python scripts to define new settings not available in Blender. The complete list of properties that users need to add and edit through the BAGEL interface is shown in Table 1. In terms of building usage, four different types are available for selection: residential, office, school and hospital. For each building type, BAGEL includes a deterministic schedule defining the amount of internal heat gains and their temporal distribution over a week period. To consider storage effects of the building mass, the thermal heaviness of the building is defined by one of the following options: light, medium and heavy. Each option corresponds to a certain value of the thermal capacitance of the building structure according to the ISO 13790. Regarding thermal characteristics of the building envelope, U-values of walls, floor, roof and windows can be assigned to each building in the Blender scene. In addition, windows have to be defined by a window-to-wall ratio, which represents the ratio of window areas to opaque wall areas, and a solar energy transmission coefficient, which is the ratio of transmitted solar radiation to incident solar radiation. Lastly, users need to provide a value for the air change per hour, which is the measure of how often the air volume is completely changed with outdoor air in one hour.

2.2.3 Modelica model generation

Once all the properties have been assigned, BAGEL is able to export Modelica models. Modelica models are stored in simple text files written in the Modelica language. These text files are constructed using a template-based approach (`https://www.makotemplates.org/`), where placeholders for the necessary simulation parameters are embedded in the file. These necessary parameters are made available by BAGEL and are then mapped to the template automatically. After mapping the parameters defined as placeholders in the Mako template, the last step performed by BAGEL is the rendering of the templates to generate the Modelica `.mo` files.

Table 1. Properties to be assigned to building objects.

Property	*Visual editing method*
Usage	Multiple choice menu
Thermal mass	Multiple choice menu
U-value walls	float number
U-value roof	float number
U-value floor	float number
U-value windows	float number
Window-to-wall ratio	float number [0:1]
G-factor	float number [0:1]
Air change rate	float number

The code generator was designed for a predefined Modelica model class representing the 5R1C model previously mentioned. This is a lumped-capacitance model where the thermal behavior of the building is described by means of an equivalent resistive-capacitive electrical network consisting of five resistances and one capacitance. Figure 2 shows the scheme of the 5R1C thermal network and its Modelica translation. Depending on the purposes and assumptions, many thermal networks have been proposed in literature, however, the ISO 13790 model is still used for its simplicity, replicability, and few requirements of input parameters. A detailed evaluation of the ISO 13790 model accuracy and limitations can be found in Vivian et al. (2017).

The thermal zone is modeled with three temperature nodes, the indoor air temperature (T_{air}), the envelope internal surface temperature (T_s) and the building mass temperature (T_m) and two boundary condition nodes, supply air temperature (T_{sup}) and the external air temperature (T_e). The five resistances are related to heat transfer by ventilation (H_{ve}), windows ($H_{tr,w}$), opaque components (split

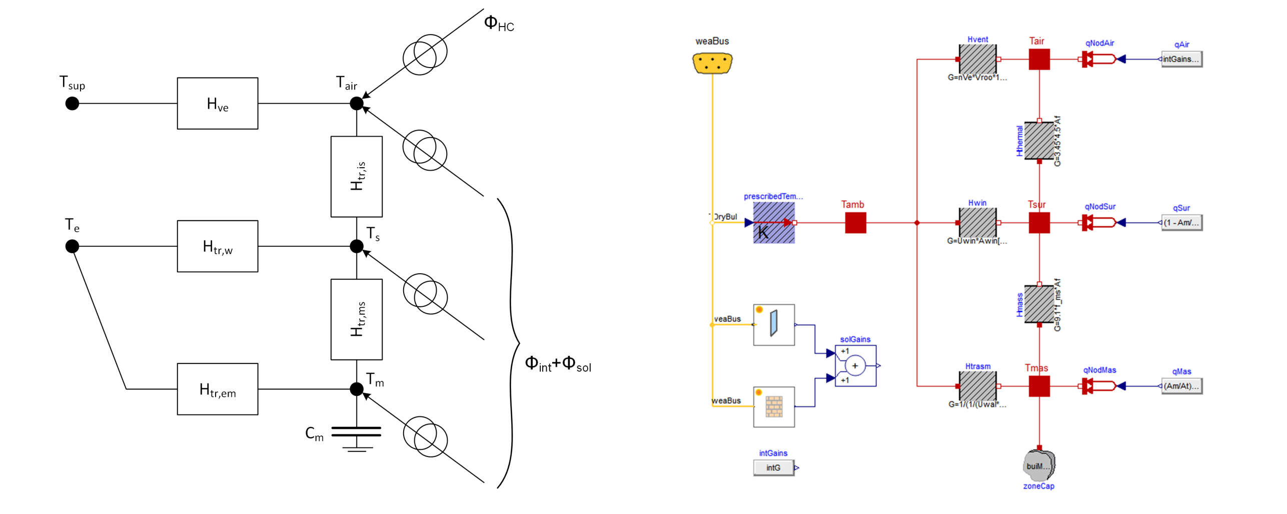


Figure 2. ISO13790 model.

between $H_{tr,em}$ and $H_{tr,ms}$) and heat transfer between the internal surfaces of walls and the air temperature ($H_{tr,is}$). The thermal capacitance C_m is connected with the temperature node T_m and includes the thermal capacity of the entire building. The heating and/or cooling demand is found by calculating the heating and/or cooling power ϕ_{HC} that needs to be supplied to, or extracted from, the internal air node to maintain a certain minimum or maximum setpoint temperature.

The following code listing demonstrates the principle upon which the Modelica model class is parameterized during the code generation process based on a Mako template.

Listing 1. Text file as Mako template

```
model ${name}
    ISO13790.ThermalZone room(
        Aroof=${Aroof},
        Uroof=${Uroof},
        ...)
end ${name}
```

The syntax for the variable substitution is the ${} construct. BAGEL reads the values assigned by the user to the properties of each building in the scene, and then replaces the construct with the correspondent value. As an example, assume that the user created a 3D building model named *office1* with a roof that has an area of 50 m^2 and a U-value of 0.3 W/m^{2}K. BAGEL will generate the following Modelica model, which can be simulated using simulation environments such as Dymola and OpenModelica.

Listing 2. Modelica code generated by BAGEL

```
model office1
    ISO13790.ThermalZone room(
        Aroof=50,
        Uroof=0.3,
        ...)
end office1
```

3 Use case

To show the capabilities of BAGEL, this chapter presents a use case taken from an ongoing research project. The aim of the project is to carry out a feasibility study of a 5th generation district heating and cooling (5GDHC) system in a new urban area located in the municipality of Køge (Denmark).

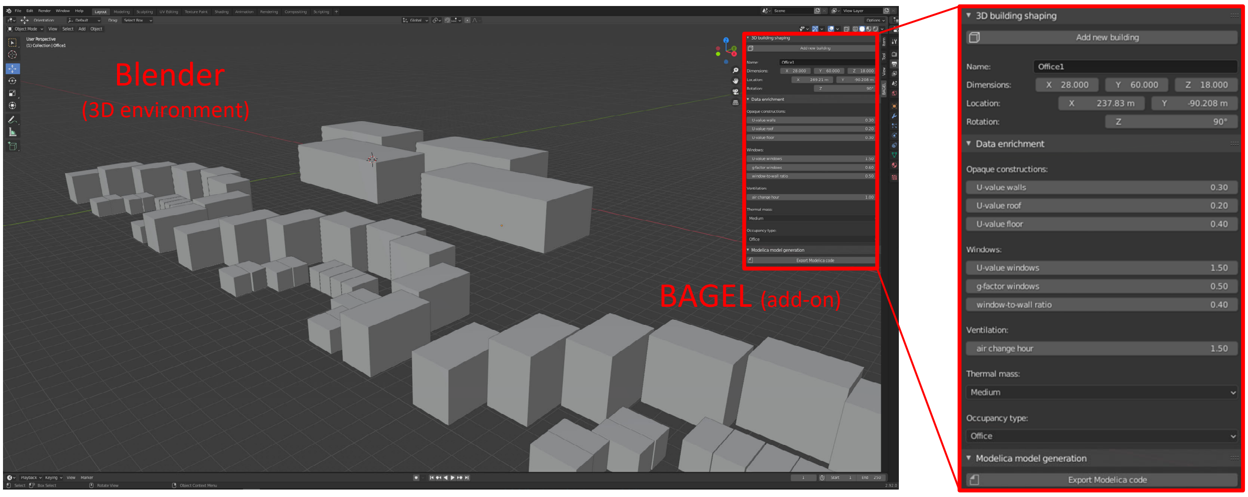
5GDHC is a recent technology that combines district heating and cooling supply into a single water network consisting of two pipes (Buffa et al. 2019). Typically, the warm pipe has temperatures of 12-20°C, while the cold pipe operates in the range 8-16°C. A particularity of 5GDHC systems is that they feature a bidirectional distribution, in which the water in each pipe segment can flow in alternating directions, depending on the net thermal fluxes in the system. Bidirectional distribution enables buildings to not only draw, but also feed heat to the network to cover heating demands of other buildings.

The efficiency and profitability of bidirectional 5GDHC systems strongly depends on the heating and cooling demand profiles of the connected buildings and their simultaneity. In particular, the efficiency of 5GDHC systems can be calculated using a metric called *Demand Overlap Coefficient (DOC)* (Wirtz et al. 2020). For discrete, equally spaced time intervals $t \in T$, the DOC of all buildings $b \in B$ in a district is defined by Equation 1:

$$DOC = \frac{2\sum_{t\in T} min\{\sum_{b\in B}\dot{Q}_{h,dem,b,t}, \sum_{b\in B}\dot{Q}_{c,dem,b,t}\}}{\sum_{t\in T}\sum_{b\in B}(\dot{Q}_{h,dem,b,t}+\dot{Q}_{c,dem,b,t})} \quad (1)$$

where $\dot{Q}_{h,dem}$is the thermal power demand for heating and $\dot{Q}_{c,dem}$ is the thermal power demand for cooling. The DOC ranges between 0 and 1. A DOC of 0 means that heating and cooling demand profiles do not overlap at all, a DOC of 1 means they match exactly. It can be noted that the District DOC is calculated solely on the basis of building energy demands.

In this context, the tool BAGEL was used to create a



Name: Office1
Dimensions: X 28.000 Y 60.000 Z 18.000
Location: X 237.83 m Y -90.208 m
Rotation: Z 90°

Opaque constructions:
U-value walls 0.30
U-value roof 0.20
U-value floor 0.40

Windows:
U-value windows 1.50
g-factor windows 0.50
window-to-wall ratio 0.40

Ventilation:
air change hour 1.50

Thermal mass:
Medium

Occupancy type:
Office

Figure 3. Blender interface with BAGEL add-on

3D model of the new urban area in Køge, visually enrich the building models with relevant properties, and export ready-to-run Modelica models. Figure 3 shows the 3D model in the Blender interface with the BAGEL add-on.

The portion of urban area considered in this use case is planned to be constructed during the next years and it includes a mix of residential and commercial buildings, as illustrated in Table 2. The required geometrical and thermal properties were assigned to the 3D building models using the BAGEL visual panel. Since this is a new urban area, and not all building information are available yet, properties were estimated according to architectural master plan, Danish building regulations and authors' assumptions. Once all properties had been assigned, Modelica files were exported by pressing the button "Export Modelica code". Then, hourly simulations were performed using Dymola. Exporting time was about 0.01 s per building, while simulation time was about 2 s per building for an one-year period. Simulations were carried out on a laptop PC with a 1.6 GHz CPU. Note that the actual location of the buildings in the scene does not affect simulation results, as buildings are completely independent from each other. Factors such as heat island effect and shading between buildings are not considered.

Table 2. Buildings details.

Typology	*No. buildings*	*Total floor area*
Single-family house	48	4320 m^2
Block apartment	32	25600 m^2
Office	4	36000 m^2

Figure 4 shows the total heating and cooling demand profiles of the urban area over an one-year period. Heating demand for domestic hot water and cooling demand for server rooms in offices were added in post-processing as constant profiles. Hourly values of total heating and cooling demand were used to calculate the DOC of the urban area by applying Equation 1. The calculation resulted in a DOC equals to 0.38. According to Wirtz et al. (2020), for DOCs larger than 0.3, a heating and cooling supply with a 5GDHC system has a higher exergy efficiency compared to a reference system. Since the calculation of the DOC is based only on heating and cooling building demand, no models related to the district energy network were needed in this case study. The actual modeling and simulation of the district energy network is planned as a next step during the project.

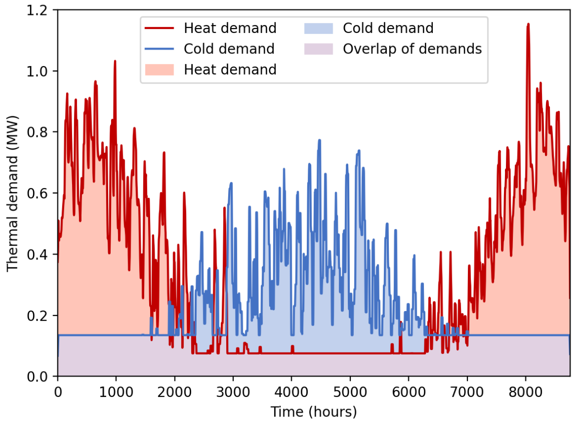


Figure 4. Heating and cooling demand profiles (daily average).

4 Conclusion and outlook

This paper presented the development of BAGEL, an open-source tool that provides automatic conversion of 3D building models into Modelica models for urban energy simulations. The software is written in Python programming language and it is developed as an add-on for the 3D creation suite Blender. BAGEL enables users to create 3D building geometries, perform data enrichment and execute generation of ready-to-run low order Modelica building models. All these actions can be performed graphically through the BAGEL interface, which is embedded in the Blender 3D environment as a visual panel. The possibility to rapidly create 3D building geometries from scratch can be particularly useful in early design stages of new urban districts and neighborhoods, when CityGML and BIM files are not usually available.

To demonstrate the capabilities of BAGEL, a use case was introduced. A 3D enriched model of a new urban area in Denmark was developed in order to calculate heating and cooling demand. This allowed to estimate the efficiency and profitability of a 5GDHC system by simulating the generated Modelica models in Dymola.

Future developments of BAGEL will focus on the integration of archetype building models through the software TEASER. In addition, possibilities to account for mutual shading between buildings will be investigated. Moreover, visualization of simulation results within the Blender scene will be explored. The source code of BAGEL, together with the Modelica ISO 13790 model, will be released on GitHub by the end of October 2021.

Acknowledgements

This work was financially supported by the Danish Energy Agency, under the Energy Technology Development and Demonstration Program (EUDP).

We gratefully acknowledge the financial support by Federal Ministry for Economic Affairs and Energy (BMWi), promotional reference 03EWR020E (*Reallabor der Energiewende: TransUrban.NRW*).

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

Blender Online Community (2021). *Blender - a 3D modelling and rendering package*. Blender Foundation. Blender Institute, Amsterdam. URL: http://www.blender.org.

Buffa, Simone et al. (2019). “5th generation district heating and cooling systems: A review of existing cases in Europe”. In: *Renewable and Sustainable Energy Reviews* 104, pp. 504–522. DOI: 10.1016/j.rser.2018.12.059.

Hensen, Jan and Roberto Lamberts (2019). *Building Performance Simulation for Design and Operation*. Routledge. ISBN: 9781138392199.

International Standard Organization (2008). *ISO 13790:2008 Energy performance of buildings — Calculation of energy use for space heating and cooling*.

Jorissen, Filip et al. (2018). “Implementation and Verification of the IDEAS Building Energy Simulation Library”. In: *Journal of Building Performance Simulation* 11 (6), pp. 669–688. DOI: 10.1080/19401493.2018.1428361.

Kent, Brian R. (2013). “Visualizing Astronomical Data with Blender”. In: *Publications of the Astronomical Society of the Pacific* 125.928, pp. 731–748. DOI: 10.1086/671412.

Müller, Dirk et al. (2016-09). “AixLib - An Open-Source Modelica Library within the IEA-EBC Annex 60 Framework”. In: *BauSIM2016 Conference*.

Nytsch-Geusen, Christoph et al. (2012-09). “Modelica BuildingSystems - Eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme”. In: *BauSIM2012 Conference*.

Nytsch-Geusen, Christoph et al. (2019-03). “BIM2Modelica – An open source toolchain for generating and simulating thermal multi-zone building models by using structured data from BIM models”. In: *13th International Modelica Conference*, pp. 33–39. DOI: 10.3384/ecp1915733.

Open Geospatial Consortium (2012). *OGC city geography markup language (CityGML) encoding standard 2.0.0*. Tech. rep.

Pyka, Martin et al. (2010). “fMRI data visualization with BrainBlend and Blender”. In: *Neuroinformatics* 8.1, pp. 21–23. DOI: 10.1007/s12021-009-9060-3.

Remmen, Peter et al. (2018). “TEASER: an open tool for urban energy modelling of building stocks”. In: *Journal of Building Performance Simulation* 11.1, pp. 84–98. DOI: 10.1080/19401493.2017.1283539.

Reynders, Glenn et al. (2017-08). “Towards an IFC-Modelica tool facilitating model complexity selection for building energy simulation”. In: *15th IBPSA Conference*, pp. 2257–2266.

Scianna, Andrea (2013). “Building 3D GIS data models using open source software”. In: *Applied Geomatics* 5.2, pp. 119–132. DOI: 10.1007/s12518-013-0099-3.

Southall, Ryan and Filip Biljecki (2017). “The VI-Suite: a set of environmental analysis tools with geospatial data applications”. In: *Open Geospatial Data, Software and Standards* 2.23, pp. 1–13. DOI: 10.1186/s40965-017-0036-1.

Thorade, Matthis et al. (2015-09). “An open toolchain for generating Modelica code from Building Information Models”. In: *11th International Modelica Conference*, pp. 383–391. DOI: 10.3384/ecp15118383.

Vivian, Jacopo et al. (2017). “An evaluation of the suitability of lumped-capacitance models in calculating energy needs and thermal behaviour of buildings”. In: *Energy and Buildings* 150, pp. 447–465. ISSN: 0378-7788. DOI: https://doi.org/10.1016/j.enbuild.2017.06.021.

Wetter, Michael et al. (2014). “Modelica Buildings library”. In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506.

Wirtz, Marco et al. (2020). “Quantifying demand balancing in bidirectional low temperature networks”. In: *Energy and Buildings* 224. DOI: 10.1016/j.enbuild.2020.110245.

Examination of Reduced Order Building Models with Different Zoning Strategies to Simulate Larger Non-Residential Buildings Based on BIM as Single Source of Truth

David Jansen[1] Veronika Richter[2] Diego Cordoba Lopez[1] Philipp Mehrfeld[1] Jérôme Frisch[2]
Dirk Müller[1] Christoph van Treeck[2]

[1]Institute for Energy Efficient Buildings and Indoor Climate, E.ON Energy Research Center, RWTH Aachen University, Germany, {david.jansen, diego.cordoba, pmehrfeld, dmueller}@eonerc.rwth-aachen.de
[2]Institute of Energy Efficiency and Sustainable Building E3D, RWTH Aachen University, Germany, {richter, frisch, treeck}@e3d.rwth-aachen.de

Abstract

Non-residential buildings are accountable for 11 % of global energy-related CO2 emissions (United Nations Environment Programme 2018). To increase the performance in this sector, Building Energy Performance Simulation (BEPS) is one feasible approach. Therefore, there is need for reliable and fast simulation models. One feasible approach are so called Reduced Order Models (ROMs). Thus in this paper, a comparison between the results of the established BEPS tool EnergyPlus and a ROM in Modelica with a reduced number of resistances and capacities is applied at the use case of a non-residential building. A self-developed toolchain was used to create equal models for ROM and EnergyPlus based on the same Building Information Modeling (BIM) model. The comparison shows that the reduced model deviates by ±10 % in annual heating and cooling. To increase accuracy and decrease computational effort the zoning strategy of non-residential buildings is investigated. The investigation shows that using a suitable zoning approach can reduce the computational effort by up to 97 %.

Keywords: BEPS, BIM, zoning, reduced order, ROM

1 Introduction

The simulation of larger non-residential buildings is an important aspect in the field of building simulations but comes with additional challenges, compared to the simulation of smaller buildings like single-family houses. Three major challenges are:

(i) Higher effort for creation and parametrization of the simulation model

(ii) Higher computational effort to solve the resulting system of equations

(iii) Necessity of zoning the simulation model due to higher influences of different room usages compared to residential buildings

Challenges (i) and (ii) can be addressed by using simplified ROMs combined with statistical data enrichment. One tool that offers these features is TEASER (Remmen et al. 2018) which provides the capability to create Modelica models based on Python code and the open-source Modelica library AixLib (Müller et al. 2016). By using Modelica, the resulting simulation models provide a huge amount of flexibility to integrate new functions and coupling the resulting BEPS models with Heating, Ventilation and Air Conditioning (HVAC) models. Challenge (iii) can be addressed by abiding existing rules of thumb but this often leads to more effort due to time consuming manual operations. The general problem that comes with zoning in BEPS is that falsely zoned models can have localized unrealistic peaks in heating and/or cooling demand. This happens due to over-discretization of zones, where e.g. the irradiated heat of high solar gains is not correctly passed to nearby zones. (Dogan, Reinhart, and Michalatos 2016; Smith, Bernhardt, and Jezyk 2011)

This paper focuses on two questions: first, in which cases using the ROM approach is suitable for simulating larger non-residential buildings, and second, how different zoning strategies affect the accuracy and computational effort. To investigate the first question, two simulation models of the same building are created by using BIM as a Single Source of Truth (SSOC) to guarantee the similarity of the models. One model is a ROM model based on the German guideline VDI 6007 -1 (2015) created with TEASER, the other is an EnergyPlus simulation model. The results of these models are compared and put into the context of previous research results. The second question is answered by applying different levels of zone reduction on the TEASER model based on existing research and investigating the results. Thereby, the special case of the used multi-zone model which uses adiabatic inner walls is also taken into account.

DOI
10.3384/ecp21181665

2 Related Work

2.1 Model Comparison

Initial verification of the ROM was already done by Lauster, Constantin, and Remmen (2017) based on the ASHRAE 140 (ASHRAE 2017) test cases. The same test cases were also simulated with EnergyPlus (Henninger, Robert H. and Witte, Michael J. 2015). These two studies showed that both the EnergyPlus model and the ROM are delivering results that are inside the boundaries for most of the test cases. In comparison to the other simulation tools, EnergyPlus tends to predict comparatively low annual heating demands. The ROM tends to predict too high fluctuations in the indoor temperatures and an underestimation of thermal mass for low mass buildings and vice versa an overestimation and too damped behavior for heavy mass buildings (Lauster, Constantin, and Remmen 2017). Kuniyoshi, Kramer, and Lindauer (2018) performed a comparison of EnergyPlus and a 6 resistance and 4 capacities model which uses different capacities for zone air, inner walls, outer components, and floors for a single-family house. They found that the default VDI 6007-1 model performs well for representing the conductive and convective heat transfer but has problems with the correct representation of solar irradiation. This challenge was met by the usage of a curve fitting approach instead of an area-weighted approach for the determination of the distribution factors for internal solar gains. All the listed works deal with test cases for buildings in the size of one room up to single-family houses. Therefore in this paper, an investigation of a four-storey non-resident building with mainly office usage is performed.

2.2 Zoning in BEPS

The German standard DIN V 18599-1 (2018) provides guideline values for the zoning of buildings divided into three aspects. (i) The usage type of the respective zone, (ii) the type of conditioning, and (iii) the glazing ratio, where a distinction is made between 25 % and 75 % glazing ratio. The impact of different zoning strategies was investigated by Brès et al. (2017) by applying different zoning strategies on multiple floor plans but not whole buildings. They covered perimeter and core distinctions, usage type distinctions, orientations of the zones, and finally combined these strategies. They conclude that the zoning affects the simulation results in a wide range based on used strategy and building. Especially simple approaches like perimeter core distinctions resulted in deviations up to 30 %, whereas the combination of different zoning strategies led to < 10 % of deviations in total heating load. Additional to research regarding how to zone buildings and how this affects the simulation results, multiple studies were performed regarding automatic zoning. E.g. Dogan, Reinhart, and Michalatos (2016) as well as Smith, Bernhardt, and Jezyk (2011) presented automatic approaches for zoning for complex building shapes based on algorithms with in-depth analysis. However, both approaches are valid only for the early concept phase of the building, where interior space divisions are still undefined. Another approach was developed by Georgescu, Eisenhower, and Mezic (2012). They used the Koopman operator to analyze the temperature behavior in different rooms during building simulation to carry out optimal zoning strategies. The approach shows promising results but a prior simulation of the detailed and not zoned model is mandatory. The performed test cases in the work carried out some guidelines which confirm the already mentioned rules and add the additional rule that small volume and surface areas can be merged to much larger adjacent zones with little loss of accuracy.

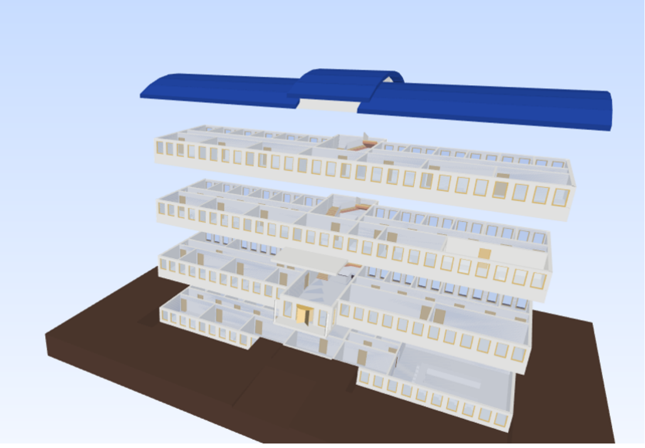


Figure 1. IFC of KIT Office Building fig. 1

Both, the comparison of ROMs and established simulation tools, as well as general investigation of zoning on simulations models, were already discussed in existing research. The special cases addressed by this paper are large non-residential buildings and the zoning of reduced ROMs using BIM models as a source.

3 Methodology

3.1 Defining the Model Setup

TEASER and EnergyPlus both in their core are based on a resistance-capacity-based approach to represent the underlying physical concepts (U.S. Department of Energy 2020; Remmen et al. 2018). However, EnergyPlus uses a much more detailed approach, while TEASER uses a reduced order approach. As EnergyPlus is an established simulation tool, the underlying assumptions will not be discussed in detail and the reader is referred to U.S. Department of Energy (2020) for further details. While TEASER is capable of exporting different types of models in this paper the two capacity model based on VDI 6007 -1 (2015) is used. Compared to the original guideline model it extended by an additional resistance for windows. Lauster, Müller, and Nytsch-Geusen (2018) already showed that this configuration is predicting the thermal behavior well for residential buildings. The periodic penetration depth that defines which part of the wall will be used as capacity and resistance and which part will be used only as resistance is set to five days according to Lauster, Müller, and Nytsch-Geusen (2018). All simulations were performed in *Dymola* using the *DASSL* solver with a tolerance of $10E^{-4}$. Both models use the test reference year of 2012 for Aachen, Germany (Lawrie and Crawley 2019).

3.2 Use Case

The investigations in this paper are applied to the Industry Foundation Classes (IFC) file KIT Office Building (Haefele, Karl-Heinz 2021) shown in Figure 1. This model is chosen because it represents a non-residential building with more complex geometry and different types of rooms, such as conference rooms, single and group offices, and laboratories. The model was created by *ArchiCAD 20* and offers good quality regarding semantic data and especially regarding 2nd level Space Boundaries (SB). Even if the used IFC file provides a comparatively good quality regarding semantics and SB, the thermal properties, layer structure of the building elements and the occupancy-related information is not completely present. To overcome this and to guarantee that both models, EnergyPlus and the ROM, rely on the same data, the building information is enriched by existing data. This data is based on the templates stored in the current release of TEASER[1].

For building physics related data, the physics for a typical building with a construction year between 1995 and 2015 and a light building structure is assumed. For occupancy-related information, the room names in the IFC are used to identify the occupancy type. Based on these types and the mentioned templates, the required information for the simulation is set. Additionally, the conditions displayed in Table 1 are applied. A constant infiltration rate (n_{inf}) and the same solar absorption coefficient ($\alpha_{solar,abs}$) for the materials are used.

For the comparison between EnergyPlus and the ROM, no internal gains are applied to reduce the influences of simulation software-related interpretation of the internal loads and thereby focus on the comparison of the simulated building physics.

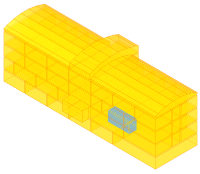
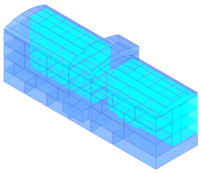
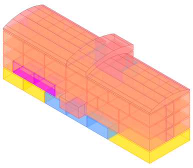
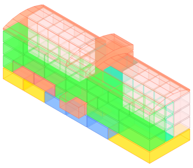
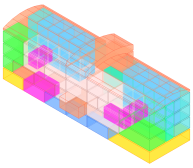
Table 1. Additional conditions.

$T_{set,cooling}$ [°C]	$T_{set,heating}$ [°C]	n_{inf} [1/h]	$\alpha_{solar,abs}$
25	20	0.2	0.7

Table 2. Four criteria for zoning.

Perimeter/Core (PC)	Internal zone External zone
Orientation (O)	North/East South/West
Glazing Ratio (GR) [%]	< 30 $30 < GR < 50$ $50 < GR < 70$ > 70
Usage (U)	*

Table 3. Zoning Setups.

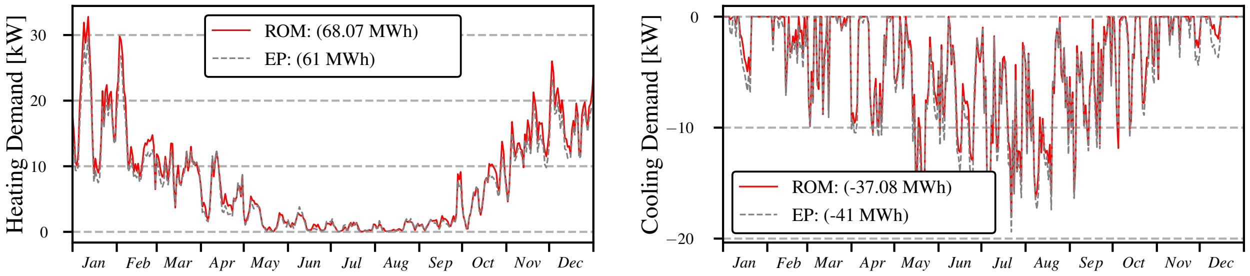
Approach	n_{Zones}	**Zoning**
PC	2	
PC + O	3	
U	6	
PC + O + U	9	
PC + O + U + GR	12	

3.3 Comparing TEASER and EnergyPlus

Using BIM as SSOC allows using different tools and approaches to create simulation models that reflect the same building.

In a suitable modeled BIM model for BEPS, all building physics and almost all simulation relevant information including profiles for internal gains can be included by using the non-proprietary IFC format (buildingSmart 2021). As the conversion of the BIM model to simulation model in terms of BEPS is still a subject of research, the self-developed toolchain BIM2SIM (Jansen, David et al. 2021) for creating simulation models of different domains is used in this paper. Besides the creation of Computational Fluid Dynamics (CFD) and HVAC simulation models, it allows also BEPS simulations with the tool TEASER (Remmen et al. 2018) and EnergyPlus (EnergyPlus 2020). As even BIM-Models created in the re-

[1] https://github.com/RWTH-EBC/TEASER/tree/development commit `95243d4`



(a) Annual heating demand

(b) Annual cooling demand

Figure 2. Annual heating (a) and cooling (b) demands.

search context are still lacking some of the information needed for BEPS, enrichment methods of BIM2SIM are used to create enriched simulation models. Using the combination of the BIM model as a SSOC and the modular toolchain, BIM2SIM allows creating twin models with the same parametrization and same boundary conditions in TEASER and EnergyPlus. As TEASER exports a Modelica multi-zone model in which all thermal zones have adiabatic inner walls, no inter-zonal heat transfer is taken into account. To provide a valid comparison, the EnergyPlus model is configured with adiabatic inner walls and without infiltration between the zones as well. In both models, every room is represented by a single zone which leads to 82 zones in total.

3.4 Zoning

After the comparison of TEASER with EnergyPlus, in the next step different zoning strategies are applied to investigate the influence on computational effort and accuracy. The reference is the not zoned model where each of the 82 rooms is represented by one separate zone. Based on the related work, the reference case will be zoned by using different combinations of the four criteria shown in Table 2. The glazing ratio is divided into 4 groups, deviating from the recommendation of DIN 18599-1.
This results in 5 separate options to zone the building, which are displayed in Table 3 in order of increasing number of criteria and thus zones. The most detailed one is a $n_{Zones} = 12$ setup where all criteria are taken into account. To merge zones of different usage, the corresponding geometries and conditioning attributes have to be averaged. In general, this information can be divided into extensive attributes likes wall areas and air volumes, intensive attributes like temperature set points, boolean attributes, and lists, like occupancy profiles. The needed functions are implemented into the algorithms to automate the process and minimize errors. The resulting methodologies are transferred into algorithms and included in the BIM2SIM toolchain. This offers the advantage that the zoning has no more to be done manually, but can be completely automated based on the information in the corresponding BIM model and thereby integrated into the workflow.

4 Results and Discussion

4.1 Comparing TEASER and EnergyPlus

The resulting models and their simulation results are compared loosely based on the methodology of the ASHRAE 140 specifications (ASHRAE 2017).
The yearly time series data for heating and cooling, including the annual heating and cooling demands, are compared in Figure 2. Comparing the results shows that both, heating and cooling dynamics on the building level are quite similar. However, the ROM tends to slightly higher peak demands for heating and lower peak demands for cooling. Also, the annual consumption for heating of the ROM is 10 % higher and the annual consumption for cooling 10 % lower compared to EnergyPlus. Comparing these results with the existing ASHRAE 140 verifications for Energyplus indicates that EnergyPlus is on the lower end of the allowed bandwidth regarding heating (Henninger, Robert H. and Witte, Michael J. 2015). The behavior that the ROM predicts higher heating and lower cooling loads can also be found in the verification made by Lauster, Constantin, and Remmen (2017).

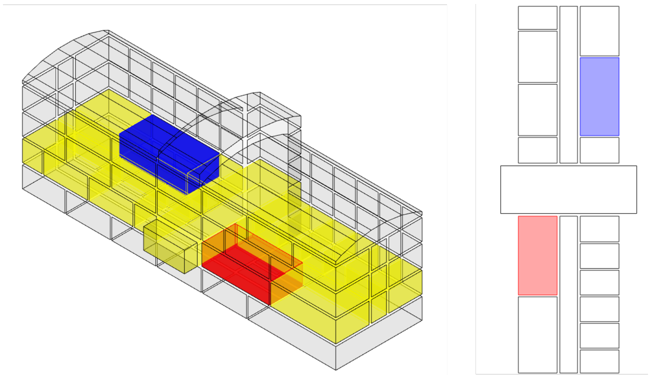
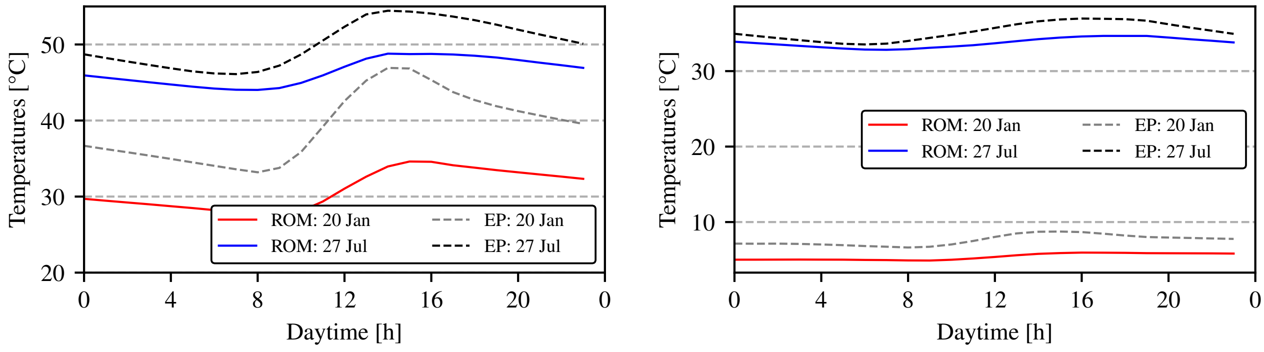


Figure 3. Selected rooms for in depth analysis.

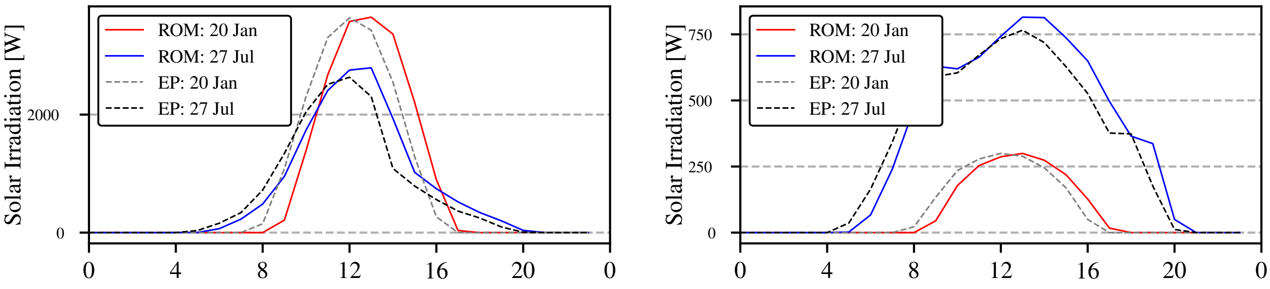
For the following investigations, two rooms are selected to be examined more closely. The rooms are highlighted in Figure 3, whereas the red one is orientated towards the south and the blue one towards the north. Both rooms have a glazing ratio $60\,\% < GR < 70\,\%$. In Figure 4 the inner temperatures of the two rooms are shown on a daily basis for a winter and a summer day, while the building is



(a) south-orientated room

(b) north-orientated room

Figure 4. Free floating temperature for two days for a south- (a) and north- (b) orientated room.



(a) south-orientated room

(b) north-orientated room

Figure 5. Solar irradiation through windows for both rooms and different days.

free-floating without heating or cooling provided. In both cases, the ROM results in lower temperatures. This behavior is more intense for the south-orientated room. The qualitative timeseries is anyhow described in the same way by EnergyPlus and ROM. The high temperatures even in winter for the south-orientated room show the effect of over-discretization for the special case of adiabatic inner walls. Due to the high glazing ratio, the temperatures in the room rise up to 48 °C in the reference case of EnergyPlus while the north-orientated room stays below 10 °C.
In Figure 5 the solar irradiation through windows for both rooms is compared for the two different days. The ROM reacts slower to the changes in irradiation as the EnergyPlus simulation does. Apart from that the solar irradation is calculated similar for both models.
In Figure 6 the heating and cooling powers for the two rooms and the same winter and summer days are shown. The time series of the ROM and EnergyPlus have the same qualitative behavior, whereby the ROM shows again the higher peak demands and reacts a bit slower. The effect of over-discretization is again visible as the south-orientated room needs to be cooled even in winter.
It was shown that the ROM predicts the annual heating and cooling in the range of ± 10 % and also the dynamic results for the whole building are deviating only slightly from the EnergyPlus predictions. Comparing these values with the literature indicates that there are large variations in simulation results even while simulating the same building. (Choi 2017) However, on a daily basis and when investigating the detailed behavior of single rooms, the ROM shows bigger deviations. Due to the assumption that both models have adiabatic inner walls, unrealistic temperatures occur in south-orientated rooms with a high glazing ratio. It can be concluded, that, especially for investigations on a daily basis or room level, the assumption of adiabatic inner walls has to be evaluated carefully.

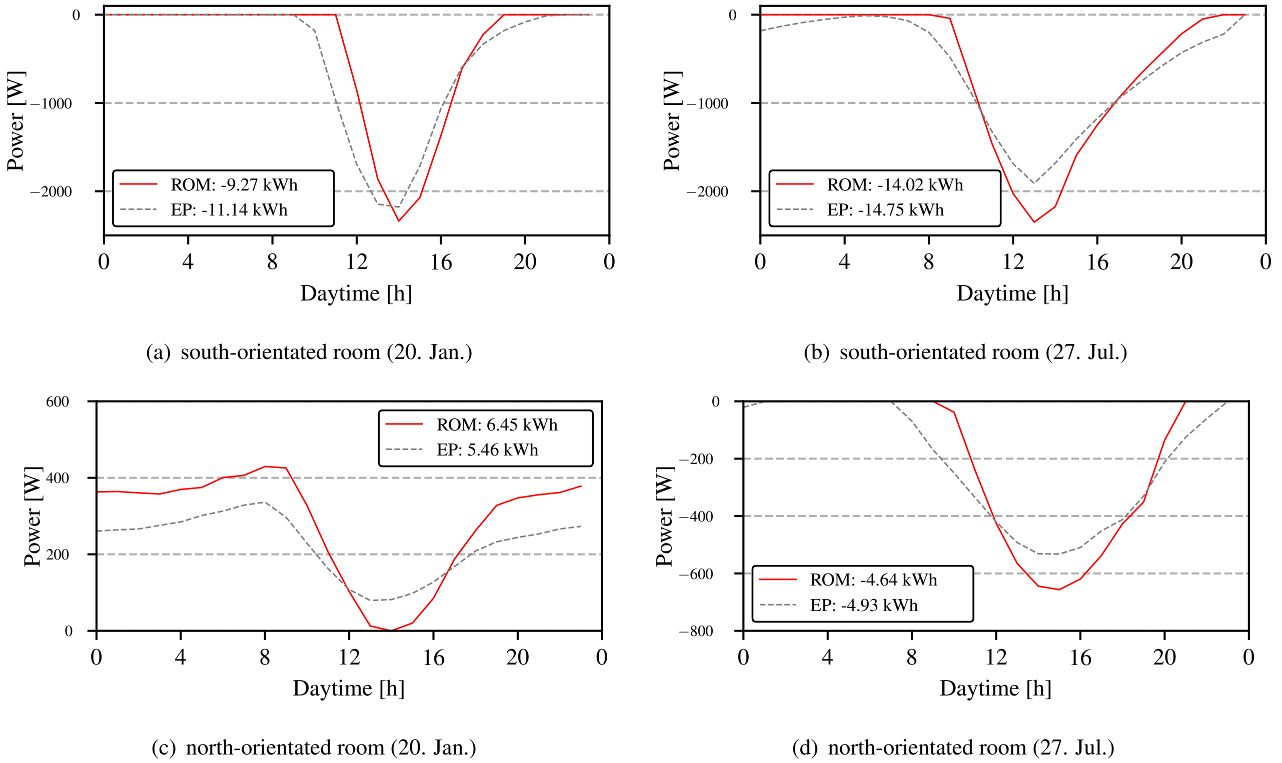
4.2 Comparing zoning strategies

The results must be evaluated concerning two questions:

(i) How to improve the computational effort without losing accuracy compared to the reference case?

(ii) How can zoning be used to reduce the influences of assuming adiabatic interior walls?

The comparison of the reference setup with $n_{Zones} = 82$ and the 5 different zoning setups introduced in Table 3 are displayed in Figure 7. The total consumption for heating and cooling is shown in total as well as the relative deviation from the reference case. Additionally, the used CPU-time for calculation is shown on a logarithmic basis. It is recognizable that especially the zoning strategies with a low number of zones have a high impact on the resulting consumption. However, the two most detailed strategies with $n_{Zones} = 9$ and $n_{Zones} = 12$ have only a deviation of around 1 % from the reference case while having a remarkable advantage in computation time. This suggests that for the use case of the here considered office building



(a) south-orientated room (20. Jan.)

(b) south-orientated room (27. Jul.)

(c) north-orientated room (20. Jan.)

(d) north-orientated room (27. Jul.)

Figure 6. Time series for heating and cooling for both rooms and different days.

the consideration of the glazing ratio does not make a decisive difference. The computational effort can be reduced by up to 97 %. It can be concluded that regarding question (i) using the $n_{Zones} = 9$ or $n_{Zones} = 12$ approach leads to a drastic reduction in computational effort while the results remain quite similar. To prove that also the dynamics are not changing, in Figure 8 the comparison between the $n_{Zones} = 9$ and $n_{Zones} = 12$ approach is shown on an annual basis.

To investigate question (ii) as a first step, EnergyPlus was used again to simulate the building but without the assumption of adiabatic inner walls. The results show that the simulated heating consumption decreases by 7.5 % and the cooling consumption by 22 % compared to the values shown in Figure 2 when the inter zonal heat transfer is taken into account. Comparing these results with the results in Figure 7 makes it clear that a smaller number of zones with a combination of south-orientated and north-orientated rooms like for the $n_{Zones} = 6$ variant, which uses only the usage for aggregation, leads to a more realistic re-

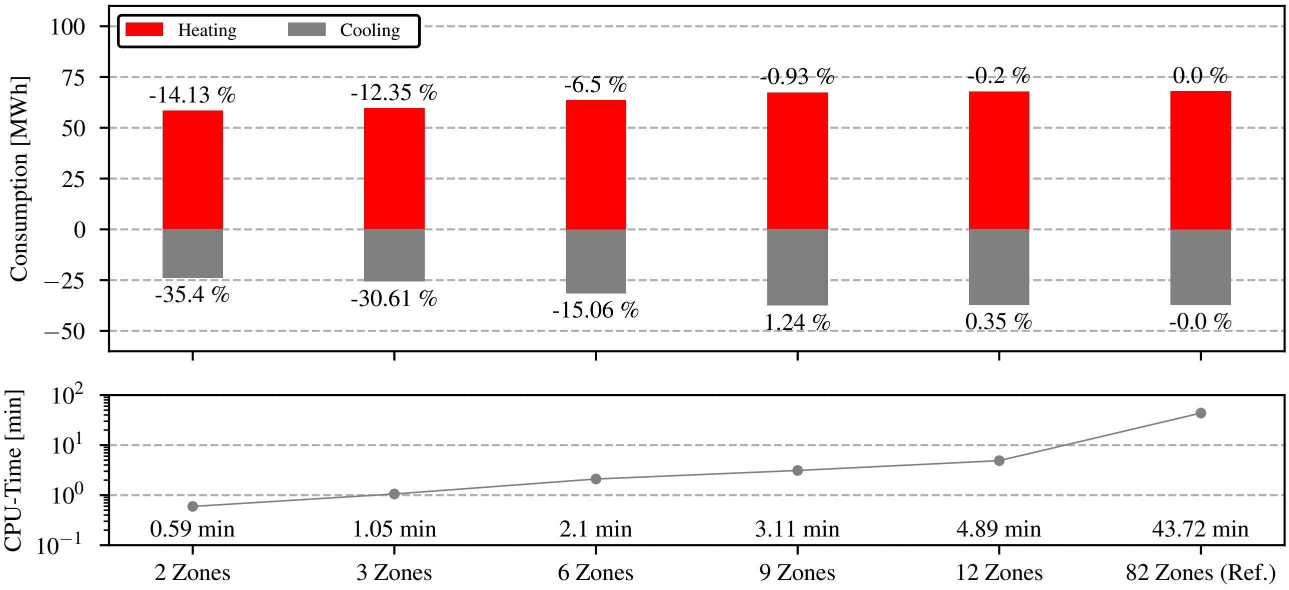
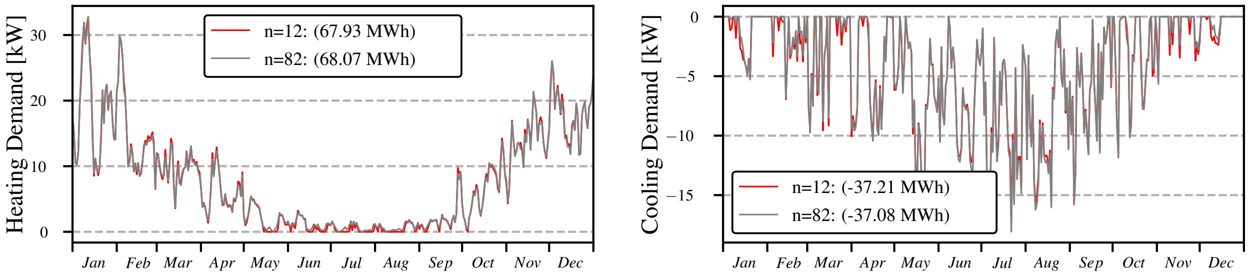


Figure 7. Consumption for the 5 zoning strategies and the reference case including relative deviation to reference case.



(a)

(b)

Figure 8. Annual heating (a) and cooling (b) demands for not zoned and zoned building with 12 zones.

sult. Therefore, it can be concluded that for ROMs, where the assumption of adiabatic inner walls is made, the most detailed approaches, where the orientation is taken into account, leads again to an over-discretization. In these cases, zoning strategies with a lower number of zones should be preferred.

5 Conclusion

The provided research performs a first investigation on comparing the simulation of complex buildings with a ROM approach against an established simulation program. By using EnergyPlus it was shown that even reduced order approaches deliver good qualitative and quantitative results and are capable to represent the overall thermal behavior of the building. Nevertheless, it was also shown that not all dynamics can be covered by the ROM. The reduced model tends to have lower temperatures and therefore higher heating and lower cooling loads, both in annual as peak demand. It can be concluded that for the investigated case of a light constructed non-residential building with a total glazing ratio of 25 % the ROM is predicting the annual heating and cooling loads, as well as the peak demands similar to the reference model in EnergyPlus within a range of 10 %. If a detailed analysis of the thermal behavior on shorter periods is wanted, one should consider using a detailed approach instead of a reduced approach. For the analysis on a yearly basis the reduced model predicts the behavior in reasonable ranges.

Furthermore, it was shown that by reducing the number of zones the computational effort can be drastically reduced compared to the not zoned building. Moreover, the reduction of the building on a really small number of zones can lead to slight to medium differences in the calculated heating loads and massive differences regarding cooling loads. For the special assumption of adiabatic inner walls used by the investigated ROM, a suiting zoning strategy can reduce the unintentional effect of too high heating and cooling demands. The investigated zoning strategies were implemented into the BIM2SIM toolchain so that they can be easily applied to new buildings in future. In future work, the comparison between TEASER and EnergyPlus should be done with internal gains and dynamic infiltration rates to cover more fluctuating changes. These changes can have different excitation frequencies which could pose an additional challenge for the reduced approach as the number of excitation frequencies that can be covered is determined by the number of capacities and resistances (**lauster_verication_2017**). Furthermore, the comparison should be extended to multiple buildings with different geometry and mass classes. Thereby, the insights of this paper can be verified. Additionally, a deeper investigation of the reasons for the over proportional increasing computational effort by the number of zones should be performed.

6 Acknowledgements

The authors gratefully acknowledge the financial support of the German Federal Ministry for Economic Affairs and Energy in the project “BIM2SIM” (project number 03ET1562A). The authors also want to thank for the support and discussions inside the IBPSA Project 1.

References

ASHRAE (2017). *ANSI/ASHRAE Standard 140:2017, Standard method of test for the evaluation of building energy analysis computer programs*. Tech. rep. Atlanta, USA: American Society of Heating, Refrigerating and Air-Conditioning Engineers.

Brès, Aurélien et al. (2017). “Impact of zoning strategies for building performance simulation”. In: *Proceedings of a meeting held 10-12 July 2017, Nottingham, UK*. Nottingham, UK.

buildingSmart (2021). *IFC4 Documentation*. *https://standards.buildingsmart.org/ifc/dev/ifc4_2/final/html/*. (Visited on 2021-05-10).

Choi, Joon-Ho (2017). “Investigation of the correlation of building energy use intensity estimated by six building performance simulation tools”. In: *Energy and Buildings* 147, pp. 14–26. ISSN: 03787788. DOI: 10.1016/j.enbuild.2017.04.078.

DIN V 18599-1 (2018). *Energetische Bewertung von Gebäuden - Teil 1: Allgemeine Bilanzierungsverfahren, Begriffe, Zonierung und Bewertung der Energieträger*.

Dogan, Timur, Christoph Reinhart, and Panagiotis Michalatos (2016). “Autozoner: an algorithm for automatic thermal zoning of buildings with unknown interior space definitions”. en. In: *Journal of Building Performance Simulation* 9.2. Number: 2, pp. 176–189. ISSN: 1940-1493, 1940-1507. DOI: 10.1080/19401493.2015.1006527.

EnergyPlus (2020). www.energyplus.net.

Georgescu, Michael, Bryan Eisenhower, and Igor Mezic (2012). “Creating zoning approximations to building energy models using the Koopman operator”. In: *IBPSA USA Sim Build*.

Haefele, Karl-Heinz (2021). *KIT IFC Examples - IfcWiki*. https://www.ifcwiki.org/index.php?title=kit_ifc_examples. (Visited on 2021-05-07).

Henninger, Robert H. and Witte, Michael J. (2015). *EnergyPlus 8.3.0-b45b06b780 Testing with Building Thermal Envelope and Fabric Load Tests from ANSI/ASHRAE Standard 140-2011*. Tech. rep.

Jansen, David et al. (2021). “BIM2SIM - Development of semi-automated methods for the generation”. In: *Building Simulations 2021 - to be published, accepted for presentation*. Bruges, Belgien.

Kuniyoshi, Ryuta, Michael Kramer, and Manuel Lindauer (2018). “Validation of RC Building Models for Applications in Energy and Demand Side Management”. en. In: *eSIM 2018 Conference Proceedings*. Montréal, Canada, p. 10.

Lauster, Moritz, Ana Constantin, and Peter Remmen (2017). “Verification and Comparison of High and Low Order Building Models from the Modelica Library AixLib using ASHRAE Standard 140”. In: *Proceedings of Building Simulation 2017*. San Francisco, USA: E.ON Energy Research Center, RWTH Aachen University.

Lauster, Moritz, Dirk Müller, and Christoph Nytsch-Geusen (2018). “Parametrierbare Gebäudemodelle für dynamische Energiebedarfsrechnungen von Stadtquartieren”. PhD thesis. Aachen. ISBN: 978-3-942789-59-2.

Lawrie, Linda K. and Drury B. Crawley (2019). *Development of Global Typical Meteorological Years (TMYx)*. http://climate.onebuilding.org.

Müller, Dirk et al. (2016). “Aixlib – An Open-Source Modelica Library Within the Iea-Ebc Annex 60 Framework”. In: *Conference Proceedings of Central European Symposium on Building Physics*. Dresden, Germany.

Remmen, Peter et al. (2018). “TEASER: an open tool for urban energy modelling of building stocks”. In: *Journal of Building Performance Simulation* 11. DOI: 10.1080/19401493.2017.1283539.

Smith, Lillian, Kyle Bernhardt, and Matthew Jezyk (2011). “Automated energy model creation for conceptual design”. In: *Proceedings of the 2011 Symposium on Simulation for Architecture and Urban Design*. SimAUD ’11. Boston, Massachusetts: Society for Computer Simulation International, pp. 13–20.

U.S. Department of Energy (2020). *EnergyPlus: Engineering Reference Version 9.4.0*. Tech. rep.

United Nations Environment Programme (2018). *Global Status Report 2018 - Towards a zero-emission, efficient and resilient buildings and construction sector*. Tech. rep. ISBN: 978-92-807-3729-5. Paris, France: International Energy Agency (IEA).

VDI 6007 -1 (2015). *Calculation of transient thermal response of rooms and buildings - Modelling of rooms*.

Accurate Robot Simulation for Industrial Manufacturing Processes using FMI and DCP Standards

Nihar Hasmukhbhai Shah[1] Perig Le Henaff[2] Clemens Schiffer[3] Martin Krammer[3]
Martin Benedikt[3]

[1]AIRBUS Operations GmbH, Germany, nihar.shah@airbus.com
[2]AIRBUS Operations S.A.S., France perig.lehenaff@airbus.com
[3]Virtual Vehicle Research GmbH, Austria,
{clemens.schiffer,martin.krammer,martin.benedikt}@v2c2.at

Abstract

Increased demand for customized products and reduced manufacturing times are key drivers towards modern, automated manufacturing systems. Manufacturing companies increasingly rely on simulation models of their manufacturing systems, with the goal to optimize critical production parameters and programming of their industrial assets. Simulation driven optimization concepts like digital twin and virtual commissioning are gaining popularity among manufacturing units to drive production rates higher. Manufacturing systems in the aerospace domain are highly complex, due to component size, tight tolerance requirements, and multi-tier manufacturing processes. Accurate simulations of robots and other programmable assets are needed, in order to lower the risk of collisions and manufacturing down times. In practice, this leads to inhomogeneous and even proprietary simulation environments, with different software interfaces. In this paper we introduce an accurate robotic arm simulation for industrial manufacturing robots that is based on open standards. This simulation environment is based on two open access standards, namely the Functional Mock-up Interface (FMI) and the Distributed Co-Simulation Protocol (DCP). In a virtualized manufacturing process the number of involved stakeholders is significantly higher. Typically, it includes software and simulation tool vendors, next to the robotic system providers. Therefore a modular software architecture based on open access standards is considered beneficial. Due to the fact that passenger aircraft are highly customized, frequent reprogramming of robotic systems is needed. During these component manufacturing processes the challenge is to maintain a high level of accuracy and reliability.

Keywords: manufacturing, robotics, co-simulation, virtualization, standards

1 Introduction

1.1 Motivation

The digitization of manufacturing progresses towards the paradigm of Industry 4.0 (Lasi et al. 2014). We can observe a strong shift from manufacturing products in a repetitive way to a significantly more smart and intelligent way of manufacturing. Especially in airplane industry, where orders are highly customized, including lots of individual adaptations, following high numbers of variants. Manufacturer Airbus produces parts across seven European countries (Mas et al. 2013) and finally assembles them to complete airplanes. The production of airplanes is massively distributed, and so is the entire supply chain behind manufacturing. Production processes are optimized for concurrency and collaboration. The goal is to enable short time-to-market and reduced cost. At the same time, quality levels should be maintained or even increased.

Today industrial robots are increasingly used in many airplane manufacturing steps. Manual offline programming of manufacturing robots for large aircraft components is a difficult task. There are several reasons for that. First of all, offline programming refers to the process of defining robotic movements when the robot is not in service. The manufacturing line has to be stopped for the programming process. Second, offline programming is constrained by numerous frame conditions. For example, space for robot movement is often limited, due to robot or machinery placement and complexity of parts. Third, inadvertent contact and collisions between the robot and airplane parts must strictly be avoided. Airplane materials and parts are expensive, and even partly manufactured composite parts are valuable. Damaged parts must be replaced, this adds up additional cost, generates waste, and slows down production. The further manufacturing processes are progressed, the more important it is to avoid damage to parts. For these reasons collision free path planning is important. In practice, even small changes to the manufacturing process may have severe impact to robot movement, and therefore programming. Therefore maximum flexibility of configuration and reconfiguration is key to speed up manufacturing and increase facility output. Finally and fourth, manual robot programming is time consuming and subject to improvement.

Due to advancements in simulation technology, virtual validation has been identified as a key method to over-

DOI
10.3384/ecp21181673

come aforementioned problems. Virtual validation refers to the solution, where a software programmed robot can be tested and evaluated in a simulated space, before the moving on to the real robot. This work focuses on the infrastructure required for virtual validation. As airplane parts are provided by a large base of suppliers and go through many different process steps, the used software tools are diverse. As a consequence, virtual validation needs to be able to deal with numerous different interfaces. Supporting many interfaces turns out to be costly.

1.2 Approach

In this work we are investigating the capabilities of FMI and DCP standards, to cope with virtual validation of the behaviour of robotic systems. FMI stands for *Functional Mock-up Interface* (Blochwitz et al. 2011). It represents a software standard for co-simulation in several industry sectors. It was proposed to solve the need for interoperability between models, solvers and tools. FMI was developed in the MODELISAR project, starting in 2008. The FMI specification is standardized as a Modelica Association Project (MAP). Its most recent specification version is 2.0.2 which was released in 2020. The FMI specification document defines an interface for model exchange and co-simulation. Today more than 100 software tools support the FMI[1]. The Distributed Co-Simulation Protocol (DCP) is an application-level communication protocol. It was designed to integrate models or real-time systems into simulation environments. It was developed in the ACOSAR project (Krammer, Marko, and Martin Benedikt 2016). It enables exchange of simulation related configuration information and data by use of an underlying transport protocol (such as UDP, TCP, or CAN). At the same time, the DCP supports the integration of tools and real-time systems from different vendors. The DCP is intended to make simulation-based workflows more efficient and reduce the overall system integration effort. It was designed with FMI compatibility in mind, i.e., it follows a master-slave communication principle, uses an aligned state machine implementing an initialization mechanism, and defines an overall integration process which is driven by standardized XML file formats. Version 1.0 of the DCP specification document was released as an open-access Modelica standard in early 2019 (Krammer, Martin Benedikt, et al. 2018; Krammer, Schuch, et al. 2019).

We aim at an accurate simulation of a universal robot (UR10) robotic arm. The UR10 is not compatible with, e.g., the RRS2 (realistic robot simulation) protocol. To overcome this issue, simulation environment provided by the robot vendor shall be used. For that purpose, a DCP master and slave pair shall be embedded into a FMU. The FMU can then be consumed by any FMI compatible robot programming software tool. The UR10 provides a virtual robot controller (VRC) through a virtual machine. Additionally, UR drivers are openly available and adopted by ROS (Robot Operating System). Its interfaces are specified in an open manner, but vendor specific. Due to the open nature of FMI and DCP, the necessary interfaces and configurations can be adapted effectively.

This paper is structured as follows. Section 2 provides an overview of related work from the fields of robotics and distributed co-simulation. In Section 3 our main contribution is stated. Section 4 highlights main results. Section 5 summarizes and concludes this paper.

2 Related Work

Realistic Robot Simulation (RRS) (Bernhardt, Schreck, and Willnow 1994; Bernhard, Schreck, and Willnow 2001; Bernhardt, Schreck, and Willnow 2001; Bernhardt, Schreck, Willnow, and Baumgartner 2002) is an initiative of automotive companies, robot manufacturers, simulator manufacturers, line builders, and measurement system manufacturers. It aims at enhancement of robot simulation accuracy and methodologies for robot off-line programming. RRS-2 defines a Virtual Robot Controller (VRC) interface. Its specification is maintained by Fraunhofer IPK and is not openly available. Only few robot manufacturers are offering VRCs compatible to the RRS-II protocol for simulation purpose. Most manufacturers have their own software solutions for realistic simulation. Unfortunately those are often not compatible to their product life cycle management (PLM) solutions.

A framework based on OPC-UA for distributed industrial robot control is shown in (Vick and Krüger 2018). It aims at virtualization on cloud systems and virtual machines, and uses the UR10 robot. The integration of simulation systems into OPC-UA networks is shown in (Reitz and Rosmann 2020). It focuses on data mapping from a simulation meta data model to an OPC-UA information model. The architecture allows for concurrent message passing between an OPC-UA server and the simulation. This allows for simulation of entire scenarios of an automotive production line.

This publication focuses on co-simulation techniques. A survey regarding the wide field of co-simulation is presented in (Gomes et al. 2018). A real-time co-simulation platform for virtual commissioning of production systems is presented in (Scheifele, Verl, and Riedel 2019). In a similar way, virtual commissioning using co-simulation for virtual plants is shown in (Süß, Strahilov, and Diedrich 2015). A co-simulation platform for design of networked control systems is shown in (W. Li, Zhang, and H. Li 2014).

In the field of co-simulation, the coupling between variables represents one of the largest challenges (Martin Benedikt and Hofer 2015). A solution to overcome the coupling challenge is presented in (Benedikt et al. 2013). In (Stettinger et al. 2014) co-simulation is extended to the real-time domain by using a model-based

[1] http://fmi-standard.org/tools/

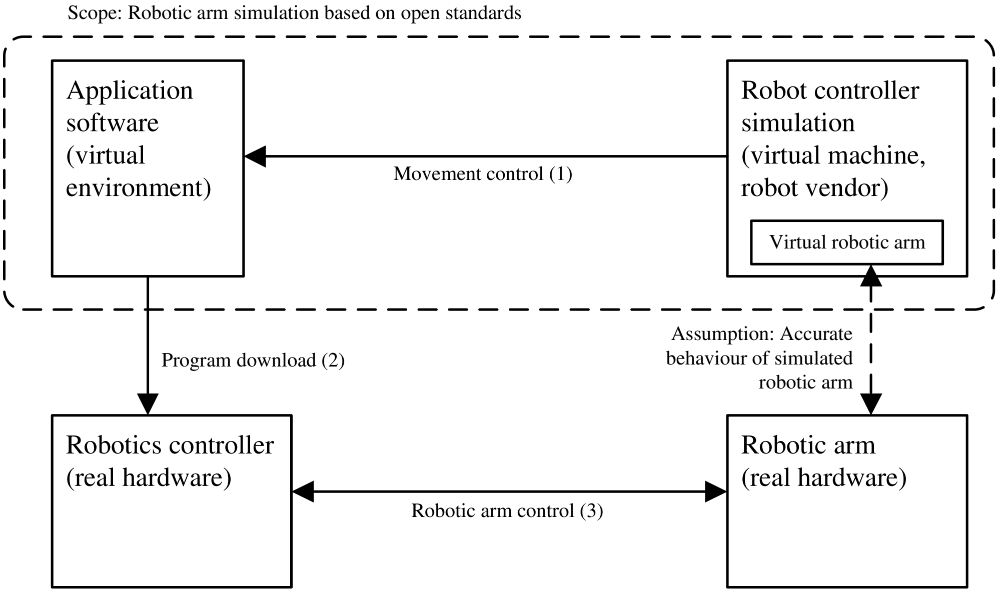

Scope: Robotic arm simulation based on open standards
Application software (virtual environment)
Robot controller simulation (virtual machine, robot vendor)
Virtual robotic arm
Movement control (1)
Program download (2)
Assumption: Accurate behaviour of simulated robotic arm
Robotics controller (real hardware)
Robotic arm (real hardware)
Robotic arm control (3)

Figure 1. Accurate simulation of a robotic arm system, including simulation based on open standards.

extrapolation scheme, to compensate round-trip time and noise-handling. A mixed real-virtual prototype from the automotive domain based on utilization of the DCP is shown in (Baumann et al. 2019). A distributed demonstrator consisting of a small scale test bed connected to a co-simulation environment is used for performance evaluation. Another example for a real-time co-simulation application can be found in (Rehtanz and Guillaud 2016).

3 Co-Simulation Architecture for Robotic Arm Control

3.1 Concept

The main concept of the proposed solution is shown in Figure 1. The application software on the upper left side provides a virtual environment for the robot and the objects it interacts with. This application software is able to integrate functional mock-up units (FMUs), hence it acts as an FMI master.

The robotics simulation on the top right hand side shall be provided by the robot vendor. This ensures the best possible simulation, having compatibility, consistency, and accuracy in mind. For the intended target robot UR10 a virtual machine is available. This virtual machine is accessible and provides an entry point for adaptations. It is intended to act as a DCP slave.

The center part of the concept is a co-simulation platform. It must be able to control one or more robot simulations, thus act as a DCP master. On the other hand, it must abstract robot connectivity behind the FMI, and act like an FMU for co-simulation.

The goal is to build an accurate and real-time capable robotic arm simulation, with a simulation infrastructure that is fully based on open standards. In a broader sense of virtual validation, the robotic arm simulation shall be used as a central component for safe and reliable robot programming. Finally, the simulation shall imitate the movement and behavior of the real hardware robotic arm, so that manufacturing processes can be planned and analyzed with the best possible predictive capabilities.

3.2 Implementation

The aforementioned concept was implemented as shown in Figure 2. The application software (3DExperience) is capable of running FMUs. To establish communication between the virtual environment and the virtual robot controller a co-simulation platform is used. There are three main characteristics of the co-simulation platform.

1. It can be accessed as an FMU from the outside.
2. It is capable of acting as a DCP master.
3. It is able to provide DCP slave functionality.

The DCP master is capable of configuring and operating a DCP simulation scenario. Technically, it will establish a configuration for two slaves. It distributes configuration information, such as variable input and output configurations. Typical parameters include step size and time resolution, as well as the network configuration information for each variable. In our architecture one slave (slave A) represents the robotic simulation. It is implemented using DCPLib. The other slave (slave B) ensures DCP access to the co-simulation platform. This summarizes the logical view on our architecture.

However, in reality the DCP master and slave B are realized in a monolithic fashion. It is able to send and receive simulation data as defined in the DCP specification.

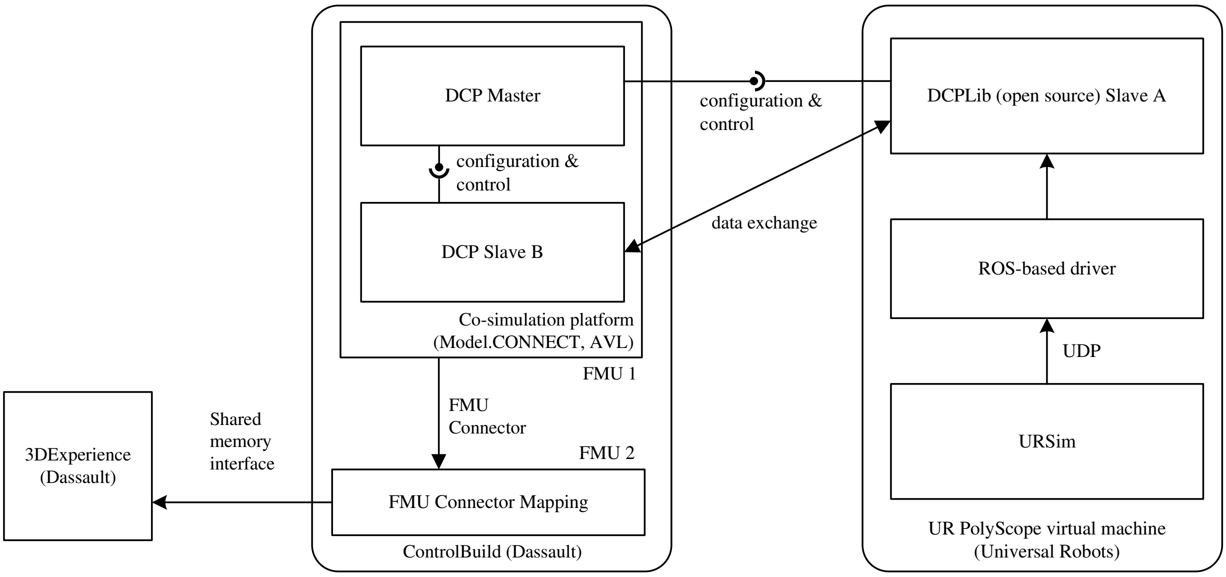


Figure 2. Realized tool couplings and interfaces to use URSim with 3DExperience

This simplifies the architecture as it is the only DCP component in the co-simulation platform FMU. If the used DCP master would not be capable of data exchange, a separate, encapsulated DCP slave (representing slave B) is needed within the FMU for this purpose.

When the FMU is instantiated by the FMI master the DCP scenario is configured and started. This results in data exchange via the DCP protocol between FMU 1 and the robotic controller simulation. This data is then relayed via FMU 2 to the application software (3DExperience) and vice versa.

3.3 Configuration Management

One of the main advantages of this approach is its impact on configuration management. Both FMI and DCP standards rely on static description files. So for each FMU a `modelDescription.xml` contains the necessary information for structural integration. In a similar way the DCP slaves, in particular Slave A, can be structurally integrated by use of a standardized DCP slave description file. The co-simulation platform consumes this DCP slave description file. After that, the co-simulation platform needs a mapping, to associate all variables from DCP to variables of FMI, and vice versa. The only code that needs to be compiled in a build process from source is the DCPLib code for Slave A.

3.4 Scalability

In many industrial manufacturing tasks multiple robots have to cooperate or interact with other manufacturing appliances. Typical examples are multiple robotic arms performing work on the same part, transportation vehicles, conveyor belts, and similar. Following the introduced simulation architecture, a larger number of DCP slaves is required to incorporate these appliances. This can be achieved in two different ways.

Assuming that one instance of a co-simulation platform is used to control one DCP slave, the application software must be capable of integrating multiple FMUs. By using this solution, the complexity of DCP communication remains at a low level. But at the same time, multiple instances of the co-simulation platform are required. This poses increased requirements to resources (memory, CPU, etc.) of the host system(s).

In contrast to this solution, the application software may continue to use one single FMU, but increase the number of exchanged variables to control more devices. This can be achieved by using array data types or multiple variables. In this case, the co-simulation platform has to register and control multiple DCP slaves. Technically, there are two options for this as well. Instantiation of one master per DCP slave, or instantiation of one single master controlling multiple slaves.

Combinations of these two possibilities are feasible. In any way, the proposed architecture is considered to be scalable, which also depends strongly on the underlying hardware resources and their configuration.

3.5 Time Regime

The virtual robot controller has an operating frequency of 125Hz. Therefore, the operating mode of the DCP scenario was set to SRT (soft real-time) (Krammer, Martin Benedikt, et al. 2018). This means that the absolute time should be synchronized with the simulated time. The DCP output step size of the virtual robot controller is configured with 0.008ms. The DCP step size is handled independently of the FMI step size. It is guaranteed that a FMI do-step call yields the most recent DCP variable value. 3DExperience acts as FMI importer. For real-time execution of the entire simulation scenario, the FMI importer must periodically call the do-step function with the same frequency and step size as used by the DCP master.

3.6 Software Tools

DCPLib is an open-source software library maintained by Modelica Association Project (MAP) DCP. Its initial version was created as a deliverable during the ITEA 3 ACOSAR project. It consists of several packages. The core library contains common classes, like constants and PDU definitions. Furthermore, master and slave packages are available, to rapidly create DCP slaves and a master to control them. DCPLib supports UDP and TCP over IPv4 transport protocols. Furthermore, it includes packages for generation and processing of ZIP- and XML-based description files. DCPLib was used for the robotics simulation part of this work.

Model.CONNECT™ is an open model integration and co-simulation platform from AVL GmbH. Typically, it improves development efficiency by interlinking simulation models into a consistent virtual prototype. The origin of simulation models is arbitrary, as the architecture of Model.CONNECT™supports the integration of up to 40 different modeling tools. Next to these well-known modeling and simulation tools, Model.CONNECT™supports a number of open standards, including Modelica Association's FMI, DCP, and SSP specifications. This allows for pure virtual and also mixed real-virtual prototypes, based on open standards. Model.CONNECT™was used as a co-simulation platform because it already supports all necessary interfaces (like acting as a DCP-master for DCP-slaves) and a Model.CONNECT™model can be exported as an FMU to allow the interaction with any tool that can import Co-Simulation FMUs.

ControlBuild is a software platform by Dassault Systemes. ControlBuild is an open automation software platform that allows seamless progress through all phases of the application development cycle – from definition and validation of specification to implementation and deployment (Systemes n.d.). ControlBuild Validation allows virtualization of physical industrial installations. A large part of the tests is traditionally carried out on-site. In the following integration phase tests are simulated on a test platform in a near-real-life environment. ControlBuild is part of the 3DExperience portfolio and provides seamless interfaces with the 3DExperience platform for simulation and validation purposes. Based on a model-driven approach and supported by a structured set of libraries, ControlBuild is used to efficiently model, simulate, test, validate and deploy control applications according to IEC61131-3 (Programmable controllers - Part 3: Programming languages). Furthermore, ControlBuild allows co-simulation of virtualized models of different industrial assets. It is able to link their behaviour to corresponding digital representations in 3DExperience through standardized interfaces, such as FMI.

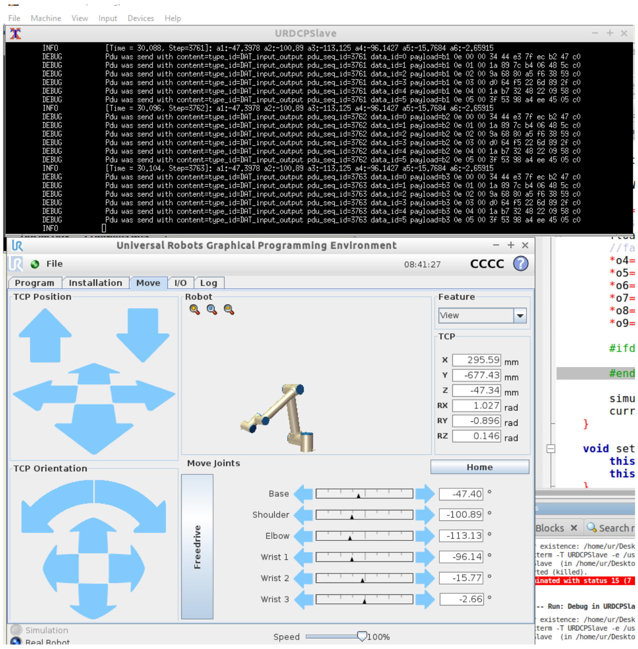
3DExperience 3DExperience is a cloud-based collaborative Product Lifecycle Management platform by Dassault Systemes. It contains software solutions for all phases of the product life cycle supporting the digital design and development of products that are subsequently manufactured. As a platform, it houses multiple capabilities (applications or apps) in a single seamless piece of software. DELMIA (Digital Enterprise Lean Manufacturing Interactive Application) is part of the 3DExperience platform that provides specialized solutions for digital manufacturing and simulations. 3DExperince supports standardized simulation interfaces, such as FMI.

4 Results

The introduced concept was implemented, including the described tool couplings and interfaces. Figure 3 shows a screen capture of the running robot simulation. On the upper side of the image the debugging output of DCPLib can be seen. `DAT_input_output` PDUs are sent, they contain a payload carrying `float64` data type values. In the rolled out DCP configuration one `data_id` per variable was used. On the lower side of the image the graphical programming environment of UR10 can be seen. It shows robotic arm controls next to the status of all joints of the robotic arm.

Figure 4 shows a visualization of a sequence of movements of the robotic arm in a three-dimensional space. The trajectories were plotted by the robot's tool tip. The dashed line represents the movement trajectory as calculated by 3DExperience (3DX). It shows clear and idealized characteristics. In contrast to that, the continuous line represents the movement trajectory as calculated by the virtual robot controller (VRC). This trajectory shows some deviations compared to the trajectory of 3DExperience.

The difference between both trajectories was calculated by using a minimum-distance algorithm. For one trajectory point, the minimum distance to a line defined by the two closest points of the other trajectory was calculated. As a result the maximum deviation of the virtual robot controller's path from the 3DExperience's path was 2.793 millimeters. The standard deviation across the entire sequence of movements amounts to 0.421 millimeters, the mean value was determined to be 0.691 millimeters. URSim considers additional robotic parameters, as physical structures and materials, as well as mechanic joints and



```
INFO     [Time = 30.088, Step=3761]: a1:-47.3978 a2:-100.89 a3:-113.125 a4:-96.1427 a5:-15.7684 a6:-2.65915
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3761 data_id=0 payload=b1 0e 00 00 34 44 e3 7f ec b2 47 c0
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3761 data_id=1 payload=b1 0e 01 00 1a 89 7c b4 06 48 5c c0
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3761 data_id=2 payload=b1 0e 02 00 9a 68 80 a5 f6 38 59 c0
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3761 data_id=3 payload=b1 0e 03 00 d0 64 f5 22 6d 89 2f c0
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3761 data_id=4 payload=b1 0e 04 00 1a b7 32 48 22 09 58 c0
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3761 data_id=5 payload=b1 0e 05 00 3f 53 98 a4 ee 45 05 c0
INFO     [Time = 30.096, Step=3762]: a1:-47.3978 a2:-100.89 a3:-113.125 a4:-96.1427 a5:-15.7684 a6:-2.65915
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3762 data_id=0 payload=b2 0e 00 00 34 44 e3 7f ec b2 47 c0
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3762 data_id=1 payload=b2 0e 01 00 1a 89 7c b4 06 48 5c c0
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3762 data_id=2 payload=b2 0e 02 00 9a 68 80 a5 f6 38 59 c0
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3762 data_id=3 payload=b2 0e 03 00 d0 64 f5 22 6d 89 2f c0
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3762 data_id=4 payload=b2 0e 04 00 1a b7 32 48 22 09 58 c0
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3762 data_id=5 payload=b2 0e 05 00 3f 53 98 a4 ee 45 05 c0
INFO     [Time = 30.104, Step=3763]: a1:-47.3978 a2:-100.89 a3:-113.125 a4:-96.1427 a5:-15.7684 a6:-2.65915
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3763 data_id=0 payload=b3 0e 00 00 34 44 e3 7f ec b2 47 c0
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3763 data_id=1 payload=b3 0e 01 00 1a 89 7c b4 06 48 5c c0
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3763 data_id=2 payload=b3 0e 02 00 9a 68 80 a5 f6 38 59 c0
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3763 data_id=3 payload=b3 0e 03 00 d0 64 f5 22 6d 89 2f c0
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3763 data_id=4 payload=b3 0e 04 00 1a b7 32 48 22 09 58 c0
DEBUG    Pdu was send with content=type_id=DAT_input_output pdu_seq_id=3763 data_id=5 payload=b3 0e 05 00 3f 53 98 a4 ee 45 05 c0
INFO     
```

TCP
X 295.59 mm
Y -677.43 mm
Z -47.34 mm
RX 1.027 rad
RY -0.896 rad
RZ 0.146 rad

Base -47.40 °
Shoulder -100.89 °
Elbow -113.13 °
Wrist 1 -96.14 °
Wrist 2 -15.77 °
Wrist 3 -2.66 °

Figure 3. Universal Robot UR10 Simulation as DCP Slave A

hinges. It is expected that this leads to a more accurate simulation result, which is considered being closer to reality as the reference trajectory by 3DExperience.

5 Conclusion

In this paper we highlight a flexible concept for accurate simulation of a robotic arm. Its simulation architecture is based on open-access standards. Furthermore, it relies on open-access interface specifications and partly on open-source software.

We have successfully demonstrated the feasibility of our approach. The proposed simulation architecture is highly modular, mainly due to the use of FMI and DCP. These open-access interface and protocol specifications provide the most flexibility. This is not only true for large original equipment manufacturers, but the entire approach has the potential to (re-)align the entire supply chain of industrial robotics and manufacturing. The proposed approach poses a strong shift from custom software tools to configurable, modular, special purpose software tools for robotics and manufacturing. The added value originates from the capability to configure the involved software tools and their interfaces. The efforts spent for coding were reduced to a minimum. In our case, DCPLib was the only software package that required a build process.

Future work includes the process of scaling up the introduced solutions to full manufacturing processes or parts thereof. For example, multiple robots operating in parallel require respective collaborative solutions for simulation. This can be achieved by modification of interfaces, and parallelization of communication to distributed components. Finally, a set of virtual robots could be effectively created by multiple instantiation mechanisms.

References

Baumann, Peter et al. (2019). “Using the Distributed Co-Simulation Protocol for a Mixed Real-Virtual Prototype”. In: *Proceedings - 2019 IEEE International Conference on Mechatronics, ICM 2019*. Ilmenau, Germany: IEEE Industrial Electronics Society, pp. 440–445. ISBN: 9781538669594. DOI: 10.1109/ICMECH.2019.8722844.

Benedikt, M et al. (2013). “NEPCE-A nearly energy-preserving coupling element for weak-coupled problems and co-simulations”. In: *Computational Methods for Coupled Problems in Science and Engineering V - A Conference Celebrating the 60th Birthday of Eugenio Onate, COUPLED PROBLEMS 2013*, pp. 1021–1032. ISBN: 9788494140761.

Benedikt, Martin and Anton Hofer (2015). “Guidelines for the application of a coupling method for non-iterative co-simulation”. In: *Proceedings - 8th EUROSIM Congress on Modelling and Simulation, EUROSIM 2013*, pp. 244–249. ISBN: 9780769550732. DOI: 10.1109/EUROSIM.2013.52. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7004951.

Bernhard, Rolf, Gerhard Schreck, and Cornelius Willnow (2001). “DEVELOPMENT OF VIRTUAL ROBOT CONTROLLERS AND FUTURE TRENDS”. In: *6th IFAC Symposium on “Cost oriented Automation”*.

Bernhardt, Rolf, Gerhard Schreck, and Cornelius Willnow (1994). “The Realistic Robot Simulation (RRS) Interface”. In: *IFAC Proceedings Volumes* 27.4, pp. 321–324. ISSN: 14746670. DOI: 10.1016/s1474-6670(17)46044-7. URL: http://dx.doi.org/10.1016/S1474-6670(17)46044-7.

Bernhardt, Rolf, Gerhard Schreck, and Cornelius Willnow (2001). “Virtual Robot Controllers as Simulation Agents”. In: *Workshop on Agent-Based Simulation, SCS - The Society for Modeling and Simulation International in cooperation with ASIM - Arbeitsgemeinschaft Simulation*, pp. 1–6.

Bernhardt, Rolf, Gerhard Schreck, Cornelius Willnow, and Alan Baumgartner (2002). “Realistic Robot Simulation in Concurrent Engineering of Manufacturing Lines in Automotive Industries”. In: *Eighth ISPE INTERNATIONAL CONFERENCE ON CONCURRENT ENGINEERING : RESEARCH AND APPLICATIONS.*

Blochwitz, Torsten et al. (2011-03). “The Functional Mockup Interface for Tool independent Exchange of Simulation Models”. In: *In Proceedings of the 8th International Modelica Conference*, pp. 105–114. ISBN: 978-91-7393-096-3. DOI: 10.3384/ecp11063105.

Gomes, Cláudio et al. (2018). “Co-Simulation: A Survey”. In: *ACM Computing Surveys* 51.3, pp. 1–33. ISSN: 0360-0300. DOI: 10.1145/3179993.

Krammer, Martin, Martin Benedikt, et al. (2018). “The distributed co-simulation protocol for the integration of real-time systems and simulation environments”. In: *Simulation Series*. Vol. 50. 10, pp. 1–14. ISBN: 9781510860230. DOI: 10.22360/summersim.2018.scsc.001. URL: https://dl.acm.org/citation.cfm?id=3275383.

Krammer, Martin, Nadja Marko, and Martin Benedikt (2016). “Interfacing Real-Time Systems for Advanced Co-Simulation - The ACOSAR Approach”. In: *STAF 2016 Doctoral Symposium and Projects Showcase*. Ed. by Catherine Dubois et al. Vienna, Austria, pp. 32–39.

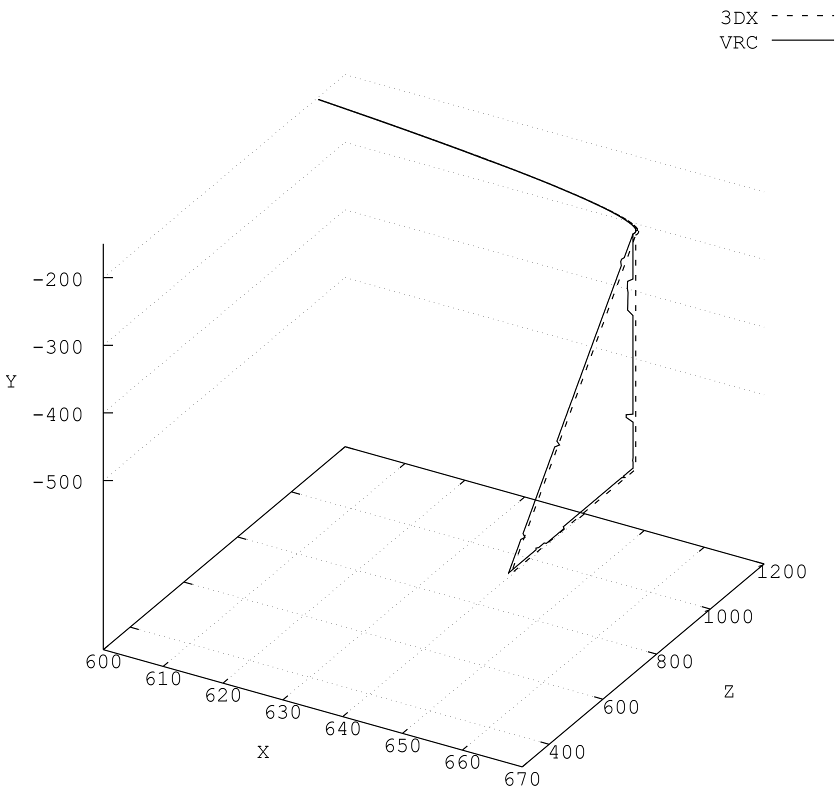


Figure 4. Visual comparison of robotic arm movement trajectories, as recorded with 3DExperience and Virtual Robot Controller

Krammer, Martin, Klaus Schuch, et al. (2019-02). "Standardized Integration of Real-Time and Non-Real-Time Systems: The Distributed Co-Simulation Protocol". In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. Vol. 157. Linköping University Electronic Press, Linköpings universitet, pp. 87–96. DOI: 10.3384/ecp1915787. URL: http://www.ep.liu.se/ecp/article.asp?issue=157%7B%5C%%7D26article=9.

Lasi, Heiner et al. (2014). "Industry 4.0". In: *Business and Information Systems Engineering* 6.4, pp. 239–242. ISSN: 18670202. DOI: 10.1007/s12599-014-0334-4.

Li, Weilin, Xiaobin Zhang, and Huimin Li (2014). "Co-simulation platforms for co-design of networked control systems: An overview". In: *Control Engineering Practice* 23.1, pp. 44–56. ISSN: 09670661. DOI: 10.1016/j.conengprac.2013.10.010. URL: http://dx.doi.org/10.1016/j.conengprac.2013.10.010.

Mas, F. et al. (2013). "Collaborative engineering: An airbus case study". In: *Procedia Engineering* 63, pp. 336–345. ISSN: 18777058. DOI: 10.1016/j.proeng.2013.08.180.

Rehtanz, Christian and Xavier Guillaud (2016). "Real-Time and Co-Simulations for the Development of Power System Monitoring , Control and Protection". In: *Power Systems Computation Conference (PSCC) 2016*. DOI: 10.1109/PSCC.2016.7541030.

Reitz, Jan and Jurgen Rosmann (2020). "Automatic Integration of Simulated Systems into OPC UA Networks". In: *IEEE International Conference on Automation Science and Engineering* 2020-August, pp. 697–702. ISSN: 21618089. DOI: 10.1109/CASE48305.2020.9216827.

Scheifele, Christian, Alexander Verl, and Oliver Riedel (2019). "Real-time co-simulation for the virtual commissioning of production systems". In: *Procedia CIRP* 79, pp. 397–402. ISSN: 22128271. DOI: 10.1016/j.procir.2019.02.104. URL: https://doi.org/10.1016/j.procir.2019.02.104.

Stettinger, Georg et al. (2014). "Model-based coupling approach for non-iterative real-time co-simulation". In: *2014 European Control Conference, ECC 2014*, pp. 2084–2089. ISBN: 9783952426913. DOI: 10.1109/ECC.2014.6862242.

Süß, Sebastian, Anton Strahilov, and Christian Diedrich (2015). "Behaviour simulation for Virtual Commissioning using co-simulation". In: *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA* 2015-October. ISSN: 19460759. DOI: 10.1109/ETFA.2015.7301427.

Systemes, Dassault (n.d.). *ControlBuild: Designing Automation and Embedded Control Systems*. online, www.3ds.com. accessed on 25 April 2021.

Vick, Axel and Jörg Krüger (2018). "Using OPC UA for distributed industrial robot control". In: *50th International Symposium on Robotics, ISR 2018*, p. 501.

Optimizing life-cycle costs for pumps and powertrains using FMI co-simulation

Miro Eklund[1,2] Jouni Savolainen[2] Antti Lukkari[3] Tommi Karhela[2]

[1]Dept. of Information Technology, Åbo Akademi, Finland, miro.eklund@abo.fi
[2]Semantum Ltd, Finland, {miro.eklund,jouni.savolainen,tommi.karhela}@semantum.fi
[3]ABB Oy, Finland, antti.lukkari@fi.abb.com

Abstract

This paper describes a collaborative digital twin approach for equipment dimensioning and selection in industrial process plants. Dynamic process simulator (Apros) was used to model the process and its automation, including pumps, while a product specific dynamic simulator (Virtual Drive) was used to model the motor and frequency converter. This approach allows all stakeholders to design and dimension the process equipment together in a holistic and energy optimal way. Simulation can be used to reach an optimal equipment solution that prevents overdimensioning, leading to up-front and total life-cycle cost savings.

Co-simulation was made possible by implementing a prototype Functional Mock-up Interface (FMI) for both Apros 6 and Virtual Drive, allowing Apros to import Virtual Drive as a Functional Mock-up Unit (FMU). This paper shows how the FMI solution can be used for finding energy optimal selections for pumps and related powertrain products.

Keywords: co-simulation, functional mock-up interface, apros, virtual drive, optimization

1 Introduction

Over 40% of the world's electricity is currently consumed by electric motors in buildings and industrial applications, and approximately 75% of these industrial motors run pumps, fans and compressors. This is a machinery category that is highly potential for major energy efficiency improvements. Considering the huge number of industrial electric motor-frequency converter systems in operation (roughly 300 million), global electricity consumption could be reduced up to 10% if these process applications were properly optimized. Thus, significant savings can be achieved in processes when using system-level optimization and right-sized components. (Waide and Brunner 2011; *Motor-driven Equipment Research Package* 2021).

Processes, systems and industrial plants are traditionally designed and dimensioned by several stakeholders. Usually an EPC (engineering, procurement and construction company) has the main responsibility of a project by handling the design, procurement and construction work. Subcontractors, such as system integrators and OEMs (original equipment manufacturers), are used to deliver the required technical systems and equipment for an industrial plant. Processes are usually divided into sub-systems, which are designed and dimensioned separately by different stakeholders. Different process equipment such as motors, frequency converters, pumps and fans are dimensioned by OEMs based on the overall specification of the system. This traditional way of designing a process is called the waterfall model, see Figure 1 for an example of such a design process. Process design challenges and approaches has been investigated in chemical engineering and process systems engineering for decades and a lot has been written about it. We will not delve deep into the available literature on the topic, but Vega et al. (2014), Nishida, Liu, and Ichikawa (1976) and Westerberg (2004) are great starting points.

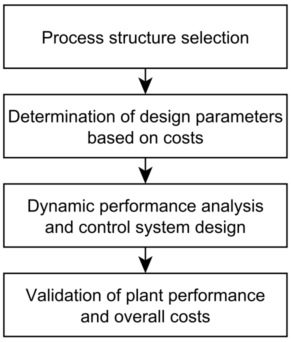

Process structure selection
Determination of design parameters based on costs
Dynamic performance analysis and control system design
Validation of plant performance and overall costs

Figure 1. A traditional way of designing a process (Vega et al. 2014).

Due to separate design steps, waterfall model lacks system-level optimization of the process. In the absence of overall coordination of the process design, each stakeholder adds their own risk margins to the sizing of the design to ensure that each and every component fulfills the critical process requirements. This leads to overdimensioning of the system components. When all these separate pieces of equipment are combined into a functional process system, such as a pumping line, the whole system runs inefficiently, using too much energy with too high costs.

Our collaborative digital twin approach combines

DOI
10.3384/ecp21181681

decision-making by including all stakeholder's equipment selections in one model. A digital twin is a virtual representation of a physical phenomenon, e.g. an industrial plant. Digital twins have been investigated as a method for improving process designs and collaboration between stakeholders, e.g. by introducing modular designs (Jiapeng et al. 2019) or by using big-data (Fei et al. 2018).

In a collaborative digital twin approach, all stakeholders design and dimension the process equipment together. With the virtual model, the behaviour and performance of the whole process can be simulated before any physical implementations. Dynamic simulation and system-level optimization allows processes to be optimized and high risk margins and overdimensioning to be avoided, while still finding suitable equipment that fulfill the critical process requirements.

In our case, system-level dynamic simulation was run using *Apros 6* (2021)® process simulator and *Virtual Drive* (2021) simulator together in co-simulation. The history of Apros goes back to the commissioning of the Loviisa nuclear power plant in Finland in the 1970s. A suitable starting point for a scientific description of Apros is Lappalainen (2019) and the references therein.

Virtual Drive is ABB's commercial software product, which can be used to model the electrical behavior of actual motor and frequency converter products in a simulation environment. Functional Mock-up Interface (FMI) was used to integrate these two simulators together and co-simulation provided reliably data about how different products would run the process. Based on the simulation results, the most optimal equipment could be selected.

This new way of working was first tested in a demo simulation in the spring of 2020. In collaboration with a pump OEM, an optimal pump-motor-frequency converter combination was dimensioned based on digital twin dynamic simulation. In the demo process model, water was pumped to a water tank which was located 20 meters above, see Figure 4. The water surface level inside the tank was attempted to be maintained the same at all times, and water outflow from the tank was constantly changed. Dynamic simulation of the system revealed that too small pump could not deliver enough water to the tank, but too big pump consumed too much energy. It was also noticed that the check valve fluttered during the simulation when using an optimal pump. After adding a feed forward structure to the process to measure the outflow rate and selecting smaller motor and frequency converter with an optimal pump, it was possible to maintain the water level in the setpoint level, and the energy consumption of the system was significantly decreased, compared to the oversized configuration. This demo confirmed the huge potential in the new way of working.

2 Methods

Implementing interoperability between two systems can be done by directly implementing the custom interface of one of the systems in the other. This means new code must be written whenever a new system needs to be added. An alternative approach is mentioned by Nouidui, Wetter, and Zuo (2014), which is to use existing standards such as the Functional Mock-up Interface. Implementing the interoperatiblity with a standardized interface like FMI gives additional compatibility with many other tools in the FMI ecosystem. In this study, a prototype FMI importer plugin was developed for Apros, allowing it to be the coordinator and import FMU models. A prototype FMI Wrapper implementation was developed for Virtual Drive, allowing Virtual Drive to be imported as an FMU model that internally uses a proxy connection to control an existing Virtual Drive instance.

2.1 Co-simulation approaches

Meer et al. (2020) describe co-simulation as typically meaning a scenario where two or more models simulate simultaneously and periodically require inputs from eachother. Time-dependent simulations also require the models to have the same concept of time, i.e. they should advance an equal amount of time between each data exchange point. The models in co-simulation can be created with different simulators, e.g. because the simulators specialize in different fields or simply because of familiarity to the modellers. Co-simulation can be implemented with local connection, see Nouidui, Wetter, and Zuo (2014) or distributed connections, see Sadjina et al. (2018). It can be implemented with custom interfaces or standardized interfaces, such as OPC Unified Architecture, e.g. Hensel et al. (2016) or the *FMI standard* (2021).

Co-simulation between Apros 6 and Virtual Drive could feasibly have been implemented in three different ways in this study. Firstly with a custom approach specific for Virtual Drive and Apros 6, secondly using OPC Unified Architecture and thirdly using FMI.

The first approach with a custom solution was ruled out immediately, as the efforts would not allow interoperability with any other systems. The second approach was more feasible, since Apros 6 already supports OPC UA, see Miettinen (2012). Further, Virtual Drive implements OPC Data Access for reading and writing some variables. Crucially though, not all variables are available in the OPC DA interface and the simulation control is implemented only with a custom interface. Thus, implementing co-simulation would require changes directly to Virtual Drive's implementation, as well as an OPC UA-to-DA conversion, such as *OPC UA Proxy* (2021).

The third approach, FMI, would enhance Apros 6 with the capability to import FMU models. For Virtual Drive, an FMI implementation would allow similar interoperability with tools in the FMI ecosystem. Further, a client-side library implementing the entirety of the Virtual Drive's custom simulation control interface existed and using that allowed rapid implementation of the FMI interface in the form of a prototype FMI Wrapper.

2.2 Functional Mock-up Interface standard

FMI is a standard that defines a set of functions, in native C, that a coordinator and a model must implement. It is supported by many tools, see *fmi-standard tools* (2021), and allows linking different simulation software and exchange of data. One software component must take the role of a coordinator that imports other software components in the form of Functional Mock-up Units (FMU models). An FMU model is essentially an archive (i.e. a .zip file), containing a modelDescription.xml file in the root, and either binary or source files. On Windows, the binaries are Dynamic Link Libraries (.dll), while on Linux they are Shared Objects (.so). The modelDescription.xml contains a list of variables, metadata about the model, FMI versions supported by the model and more. (Blochwitz et al. 2012), (Chen, Huhn, and Fritzson 2011)

Hauf et al. (2017) describe the history of the FMI standard, as well as the differences between model exchange and co-simulation sub-standards within FMI. For the purpose of this study, co-simulation was the only approach that made sense, since Virtual Drive is a complete package that contains its own mathematical calculations, essentially a black box from the point-of-view of other systems.

Functional Mock-up Interface can suffer from some weaknesses, e.g. its floating-point representation of time (Fabio et al. 2017). However, in our study this was never a problem, as the step-sizes were larger than 0.1 seconds. For such large step sizes, floating-point representation of time was accurate.

2.3 Apros 6 FMI importer

Apros 6 (2021) contains two main parts, a solver written in Fortran and a desktop application written in Java. No version of Apros 6 that has been released to date (Apros versions 6.10.x and older) comes with built-in support for importing and simulating FMU models. However, Apros 6 can be extended using *Eclipse* (2021) based plugins. In this study, a plugin was created that allows Apros 6.9 and newer versions to import and simulate FMU models that support FMI 1.0 and FMI 2.0 co-simulation.

Representing an FMU model in a simulation tool like Apros was straight forward with user components (UC). User components in Apros 6 are re-usable blocks that support scripting using the *SCL* (2021) language. Input and output signals on a UC allows data-flow between the FMU model and other components in the Apros model, while FMU parameters can be represented using UC properties. The FMI importer plugin uses *Simantics FMIL* (2021) as a Java implementation of the FMILibrary and the user component's SCL scripts import these Java functions. Due to the technical implementation of the Simantics FMIL, only x64-bit FMU models are supported in Apros. The component call flow and data flow of the plugin's auto-generated user components can be seen in Figure 2. Figure 3 shows an auto-generated Apros 6 user component from an import FMU model, as it appears in the Apros 6 model browser and diagram.

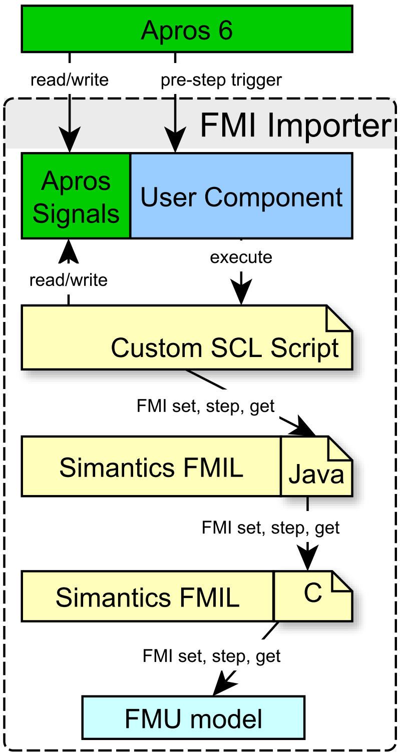


Figure 2. Component call- and dataflow for the Apros 6 FMI importer plugin. Apros 6 triggers user component's SCL scripts each step. These scripts internally use the Simantics FMIL implementation, included in the plugin. Apros signals are written to FMU models as inputs and FMU model outputs are written to Apros signals

2.4 Virtual Drive's FMI Wrapper

ABB's Virtual Drive is a Windows x86 executable that can be configured using *Drive Composer Pro* (2021) desktop application. Drive Composer Pro requires the add-on "ABB Virtual Drive" to allow creation, configuration and simulation of Virtual Drives. *DriveSize* (2021) is a product catalogue look-up tool that shows the size and specifications of ABB's drive products. These physical products can then be manually created as virtual simulation models using Drive Composer Pro and Virtual Drive. In this study, Virtual Drives were created with the specifications of a pump OEM, using DriveSize tool to find suitable equipment sizes to represent underdimensioned, overdimensioned and optimal equipment.

Virtual Drive uses the same software as physical drive equipment. By using Virtual Drive in simulation, rather than a simple modelica model of a drive and motor, the models can achieve results that reflect the real-world behaviour of drives more accurately.

FMI was possible to implement with two approaches, either directly or as a proxy. A direct approach would've

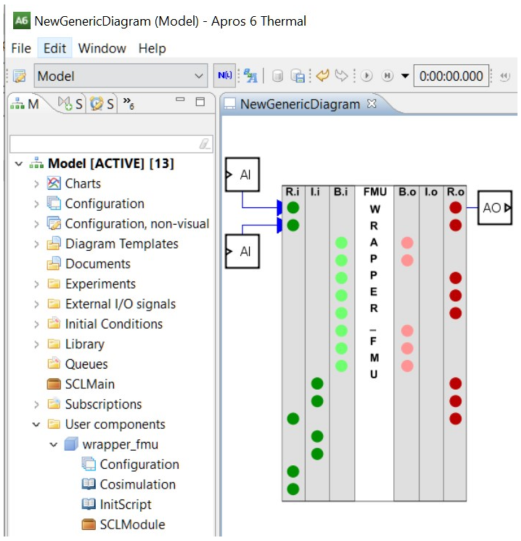


Figure 3. An auto-generated user component when importing an FMU model file using the Apros 6 FMI importer

meant packaging Virtual Drive itself as an FMU model, while a proxy approach would use an FMU model that internally uses a remote interface for communication with the Virtual Drive. The proxy approach was further divided into two sub-approaches: Tool-coupling or standalone. In a tool-coupling approach, Virtual Drives would be configured and started outside of the FMU models context and the FMU model would only connect to existing drives. A standalone approach would've meant packaging the virtual drive executable inside the FMU model and having it automatically started by the main FMU model.

The approach that was chosen was a tool-coupling proxy approach. Virtual Drive could only be compiled to x86 architecture, due to limitations in its dependencies. Thus, the direct approach would also only support x86 architecture and thus be incompatible with the Apros FMI importer plugin. A proxy approach would allow the creation of an FMU model that supports x64 and x86 architectures. Additionally, a proxy approach made sense because Virtual Drive already implemented a custom remote interface for simulation control, supported by e.g. Drive Composer Pro. A .NET client-side library implementing all of these remote function calls had been developed by ABB prior to this study, written in C#. A tool-coupling approach was chosen, as this would allow Drive Composer Pro to still be used in the configuration and monitoring of Virtual Drives. Version management would also be separated from the FMI Wrapper, allowing it to stay up-to-date with future releases of Virtual Drive and Drive Composer Pro, as long as the custom remote interface for controlling Virtual Drive does not changed.

2.5 Apros model

The process under study was modelled using the Apros simulator as it is intended for system-wide fluid process modelling. The modelled process is show schematically in Figure 4.

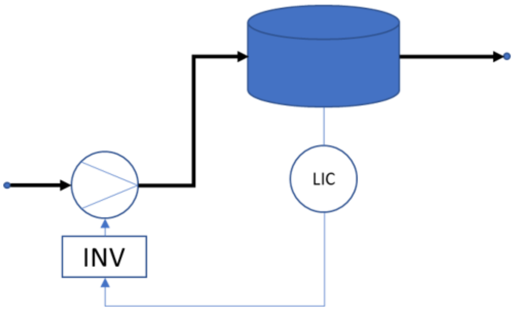


Figure 4. Schematic of the process under study, showing main process components and the liquid level control loop.

In the process we pump water to a tank, whose outlet flow varies. The goal of the level control loop (LIC) is to keep the liquid level within an allowable range of its setpoint. In the basic setting the level controller's output is the pump rotation speed setpoint. This is given to the drive+motor (INV) model who in turn returns the pump shaft torque. The process simulator then uses this value to determine the pump rotation speed and from that its head. This head value then goes to the pressure-flow solver of the simulator. In this case the fluid pressure-flow behaviour was modelled as 1D homogenous two-phase flow using dynamic conservation equations for mass, energy and momentum. The modelling was done with a graphical user interface, with no need to explicitly write the governing equations. In addition to the fluid flow components, i.e. pipes, valves, pumps and tanks, the model also included automation component. Namely, in the model we implemented a liquid level control loop. In later stages of the investigation we extended the control loop to include a feedforward term from the tank outflow measurement. To test the different drivetrain dimensionings and control structures, a simulation test sequence was used, see Figure 5.

3 Results

3.1 FMI Wrapper related results

The prototype FMI Wrapper created during this study implements the FMI 2.0 co-simulation interface. The existing client-side library, with remote function calls for Virtual Drive, had been implemented in C#, thus it was deemed the simplest solution to also write the FMI Wrapper in C# and use the existing library directly. A .NET C# project that compiles into both x86 and x64 Dynamic Link Libraries (.dll) was created, utilizing FMI c-headers

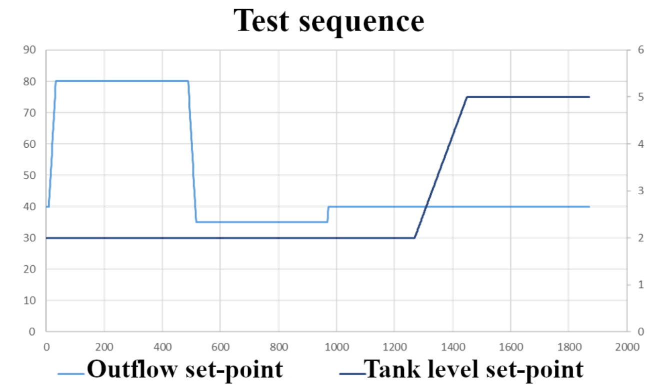


Figure 5. Simulation test sequence: Part 1: outflow variables 40 kg/s → 80 kg/s → 35 kg/s → 40 kg/s (light blue curve, primary y-axis), Part 2: tank level setpoint ramped from 2m to 5m (dark blue curve, secondary y-axis).

and the existing client-side library. Functions needed to be explicitly configured using *DllExport* (2021) for .NET like Listing 1, in order to make them compatible with native C, i.e. the FMI specifications.

Listing 1. A generic .NET DllExport usage example. All .NET projects that need to be compiled as .dll and need compatibility with native C can use this approach.

```
[DllExport("fmi2DoStep",
    CallingConvention =
        CallingConvention.Cdecl)]
public static fmi2Status fmi2DoStep(
    IntPtr c,
    double currentCommunicationPoint,
    double communicationStepSize,
    [MarshalAs(UnmanagedType.Bool) bool
        noSetFMUStatePriorToCurrentPoint])
{
    ModelInstance m = ((GCHandle.c).Target
        as ModelInstance);
}
```

Virtual Drives are given a unique local identifier when created using Drive Composer Pro. This identifier represents the local channel used to communicate with that drive instance. When importing the FMU model of the FMI Wrapper, this identifier must be set using the FMI integer-type parameter VirtualDriveLocalNodeID and it must match the identifier of an existing Virtual Drive instance that is running on the same machine.

The FMI Wrapper was tested in various simulators to confirm its implementation of the FMI 2.0 co-simulation interface was correct. The Apros 6 FMI importer plugin created during this study, FMU compliance checker, *FMI Toolbox import* (2021) and *Simulink FMI import* (2021) were successfully able to import and simulate Virtual Drive through the FMI Wrapper.

With the FMI Wrapper, Virtual Drive gained interoperability with multiple simulation tools, including Apros 6. It comes in a format that is familiar to those who already use Virtual Drives, since Drive Composer Pro is still used for configuring Virtual Drives.

3.2 Apros FMI importer related results

The Apros FMI importer went through two phases of development: In the first phase, a user component was manually created to represent the Virtual Drive's FMI Wrapper and its parameters, inputs and outputs. This was created only for the FMI Wrapper and would not have worked for any other FMU model. In the second phase, the importer was generalized to allow importing of any x64-bit FMU models implementing either the FMI 1.0 or 2.0 co-simulation interface.

A copy of the imported FMU file is stored in the auto-generated user component, which will be copied to a well-known location within the plugin's filesystem before an FMU instance is started. Exporting an Apros model that contains an auto-generated FMU user component is possible and a copy of the FMU model will be stored in the exported Apros model. Importing such a model to an Apros version without the FMI importer plugin will succeed, but simulation will fail, as the user components' SCL scripts no longer find the required Java implementation.

With the new plugin, modellers can easily import FMU models to Apros and use them as familiar user components, saving modellers' time and bringing new interoperability options to Apros.

3.3 Process engineering related results

As was describe earlier, the study consisted of simulating three pumps, each with two drivetrains. In Figure 6 we present the liquid level behaviour of three pump-motor-drive combinations.

In the figure we show three drivetrains: the green line is the so called optimal dimensioning, the yellow line is an intentionally undersized dimensioning and the brown line is an oversized case. In the chart we can also see the liquid level setpoint (blue dotted line) as well as the acceptable range for the liquid level around it (dotted yellow line and dotted grey line). The thick lines depicts the liquid level behaviour during the simulation.

The optimally dimensioned drivetrain keeps the liquid level in its range, except for a short while during the setpoint change. The underdimensioned drivetrain fails immediately and the liquid level falls to nearly zero. Finally, the overdimensioned drivetrain keeps the level closer to the setpoint than the others. This is achieved with a larger energy consumption. More specifically, the specific energy consumptions were, respectively: 0.101 kWh/t, 0.278 kWh/t and 0.13 kWh/t. These numbers reflect the utter failure of the underdimensioned drivetrain: while it is unable to achive its target, it also uses a lot of energy in doing so.

In addition, we experimented with two alternative control structures. The alternative was to add a feedforward term from the tank outlet flow measurement. This was investigated only with the optimally dimensioned case, see Figure 7.

We see that, as expected, the feedforward control nearly perfectly compensates the outflow changes. In combina-

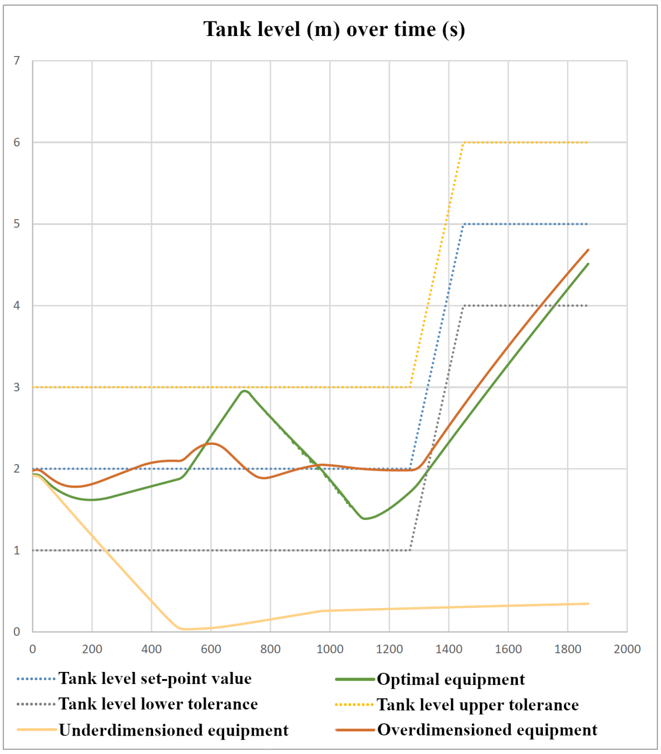


Figure 6. Liquid level behaviour with "optimal" (green line), "undersized" (yellow line) and "oversized" (brown line) drivetrains.

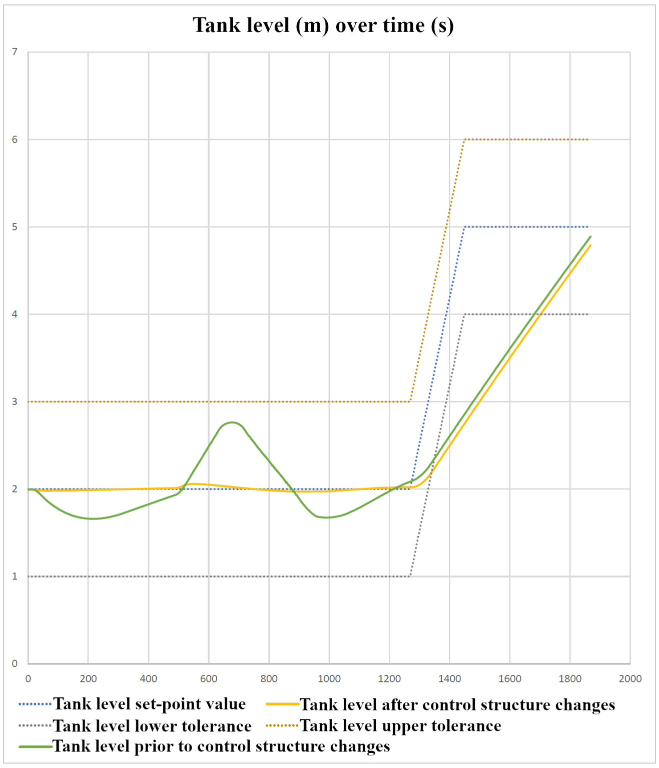


Figure 7. Liquid level behaviour with "optimal" drivetrain, using the two control structures. Green line = before changes, Yellow line = after changes.

tion with a smaller motor and frequency converter, it is able to reduce energy consumption from 0.101 kWh/t to 0.095 kWh/t, a 6% reduction. This is a significant cost and energy consumption reduction when applied to all drivetrains in a large process.

4 Analysis

4.1 FMI simulation speed

Speed was not critical during the creation of the Apros FMI importer nor for the prototype FMI Wrapper for Virtual Drive. However, Apros FMI importer plugin's speed has been compared to *Simulink 2020b* (2021) and *FMU compliance checker* (2021). The simulation speeds were tested with an example *ControlledTemperature.fmu* (2021) model, as well as the created FMI Wrapper with and without a connection to Virtual Drive. Without a connection, each FMI 2.0 function in the wrapper will simply return immediately, thus allowing us to analyse the speed loss caused purely by the Apros FMI importer. We simulated the FMU models in Apros multiple times in one-minute tests, with a step size of 0.1 and calculated the average real-time factor (RTF) for the models, with various numbers of simultaneous FMU models present in the model. In Simulink and FMU Compliance Checker, we specified 100 000.0 seconds as the target simulation time for the fastest cases, and 100.0 seconds for the slowest cases, with fixed step-size set to 0.1 seconds, and measured how long it took to simulate. The average of five executions per case was used.

The results for different simulator programs and different number of FMU instances can be seen in Table 1. The time spent loading and initializing the FMU models was not taken into account when calculating the total time, but was observed to be less than 1 second even when using 8 ControlTemperature.fmu models. This should affect the RTF factor seen in the tables positively by at most 1.7%, since 1/60 $\approx$ 1.7, and even then only for cases with 8 simultaneous models.

Clearly, the Apros FMI importer plugin is much slower that the other simulators, when using the ControlledTemperature.fmu. However, it should still be much faster with the unconnected FMI Wrapper than ControlledTemperature.fmu. Instead, it is much slower. The root cause for the slow speeds was not identified in this study and requires further investigation. An initial analysis of the speed issue would suggest that the problem is not in the Apros user component's themselves, i.e. the speed is not lost due to Apros simulation pre-step hooks triggering SCL scripts in the user components and updating signal values, but rather in the SCL functions and the underlying Java and C implementation of Simantics FMIL. Similar simulation speed issue could be seen by executing SCL scripts independently, without Apros.

As expected, connecting the Virtual Drive lead to slower simulation speeds for all simulators. In fact, the overhead introduced by the custom interface between FMI

Wrapper and Virtual Drive was so large that all simulators had at best RTF of around 3.0 for a single Virtual Drive instance, with step-size 0.1.

Table 1. Average simulation speed, as a real-time-factor (RTF), with varying number of simultaneous FMU model instances, step-size 0.1 seconds. RTF of 1.0 correspond to as fast as real-time, 2.0 means twice as fast as real-time, etc. Simulink 2020b FMU import, Apros FMI importer plugin and FMU Compliance Checker were tested. CT = ControlledTemperature.fmu, FW = FMI Wrapper without a connected Virtual Drive. FW+VD = FMI Wrapper with a connected Virtual Drive.

Simulator	Instances	CT	FW	FW+VD
Simulink	1	333	35000	3.0
-||-	2	298	25000	2.9
-||-	4	230	16000	2.5
-||-	8	136	12000	1.4
Apros	1	132	17.4	3.0
-||-	2	92.3	15.1	1.7
-||-	4	53.9	11.3	0.88
-||-	8	27.8	7.5	0.44
Checker	1	384	46000	3.2

The demonstration Apros model in this study had a different step-size than we used in the simulation speed tests. Additionally, variable step-sizes are taken, depending on what the model is currently calculating. This variable step-size is matched by the FMU instances in the model, which in turn directly affects the simulation speed. E.g. small step-sizes were taken by the Apros model when the check valve fluttered, while larger step-sizes were taken in steadier states. In the simulation speed tests, we forced the step-size to always be 0.1 seconds and the model contained only the FMU models.

4.2 Apros FMI importer limitations

Due to the usage of Simantics FMIL as the base implementation for the FMI, x86 FMU models could not be imported in x64 desktop environments, e.g. in Apros 6 desktop. Further limitations inherited from this base implementation were a lack of model exchange support. However, both FMI 1.0 and 2.0 co-simulation standards were implemented.

Unimplemented functions for FMI 2.0 were all model exchange functions, as well as:

- fmi2SetRealInputDerivatives
- fmi2GetRealOutputDerivatives
- fmi2CancelStep
- fmi2GetDirectionalDerivative
- fmi2DeSerializeFMUstate
- fmi2SerializeFMUstate
- fmi2SerializedFMUstateSize
- fmi2FreeFMUstate
- fmi2SetFMUstate
- fmi2GetFMUstate
- fmi2Reset

4.3 FMI Wrapper limitations

The prototype FMI Wrapper for Virtual Drive was implemented for both x86 and x64 architectures. The wrapper's FMU model is only available for Windows operating systems, just like Virtual Drive. The Virtual Drive and the FMI Wrapper must also physically be located on the same machine, as the remote interface requires local visibility. It is a remote interface only in the sense that the Virtual Drive executable can be started by another program than the program that imports the FMI Wrapper's FMU model.

Most configuration of Virtual Drives must still be performed using Drive Composer Pro's user interface, e.g. selection of motor and parametrization of the drive's size. Most output variables can also only be monitored using Drive Composer Pro. The prototype FMI Wrapper only exposes a small sub-selection of variables, such as torque as output, and motor speed as input. Thus, the FMI wrapper in its current state is far from standalone, since users are unable to define new Virtual Drive instances exclusively using the FMU model and the variables exposed by it.

Maximum simulation speed continues to be a challenge, due to time spent by functions in Virtual Drive and time spent by the custom remote interface between FMI Wrapper and Virtual Drive. FMI Wrapper was only implemented for the FMI 2.0 co-simulation interface, but not all functions were implemented. Virtual Drive did not support saving or loading the internal state, thus state-related functions could not be implemented.

Unimplemented functions for FMI 2.0 were all model exchange functions, as well as:

- fmi2SetRealInputDerivatives
- fmi2GetRealOutputDerivatives
- fmi2CancelStep
- fmi2GetDirectionalDerivative
- fmi2DeSerializeFMUstate
- fmi2SerializeFMUstate
- fmi2SerializedFMUstateSize
- fmi2FreeFMUstate
- fmi2SetFMUstate
- fmi2GetFMUstate

5 Discussion

Implementing the prototype FMI Wrapper was remarkably simple for Virtual Drive, since there already existed a complete client-side implementation of Virtual Drive's custom simulation interface in a C# library. An initial challenge had to be overcome, namely the C#-to-native-C function exports, but mechanisms for dealing with that were found.

The FMI Wrapper inherited the limitations of the existing Virtual Drive remote control interface, e.g. a lack of storing the internal state of Virtual Drive in a serializable blob and only having visibility to a small sub-selection of variables. The prototype FMI Wrapper developed requires Drive Composer Pro to be used for the creation and configuration of Virtual Drives.

5.1 Future Work

Apros FMI importer and the base Simantics FMIL will need a thorough investigation to reduce the overhead from using the Java FMI implementation. Additionally, parallel execution of FMU models within an Apros model, rather than serial, will reduce simulation speed losses caused by simulating multiple simultaneous FMU models.

The interoperability of Apros 6 and Virtual Drive was only tested with a simple demonstration model during this study. Future work should include validation of this approach using a real customer case. A collaborative digital twin, using Apros 6 dynamic process simulation in co-simulation with Virtual Drive, should be created to find the optimal equipment selection of a real-life process. Other ABB software and pump OEMs equipment can be used for a more detailed collaborative digital twin, assuming their simulation software is compatible with the Functional Mock-up Interface. Implementing FMI compatibility from scratch to these equipment simulators will be needed as part of future real customer cases, if they do not already implement it.

5.2 Conclusion

Traditional process design uses a waterfall model for equipment selection, where multiple sub-contractors add too high risk margins to the sizing of their equipment in order to ensure components fulfill critical requirements. In this paper we presented the first steps towards a collaborative digital twin approach as an alternative way of designing a process. This approach aims to remove overdimensioning, leading to up-front cost savings and total life-cycle cost savings for the whole process.

The approach was demonstrated with a Virtual Drive instance connected to a simple Apros model that had a few equipment selection options. However, the approach should be possible to utilize when optimizing the equipment selection in an entire industrial process plant. The approach was made possible with interoperability between simulators and we have presented Functional Mock-up Interface as a candidate for achieving this interoperability.

Acknowledgements

This activity has received funding from the European Institute of Innovation and Technology (EIT). This body of the European Union receives support from the European Union's Horizon 2020 research and innovation programme.

The authors would like to thank Reino Ruusu for their support implementing the prototype Apros FMI importer. The authors would also like to thank Daniel Rogoz and Roman Okolovich for their support implementing the prototype Virtual Drive FMI Wrapper.

References

Apros 6 (2021). URL: https://www.apros.fi/ (visited on 2021-04-23).

Blochwitz, T. et al. (2012-09). "Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models". In: DOI: 10.3384/ecp12076173.

Chen, Wuzhu, Michaela Huhn, and Peter Fritzson (2011). "A Generic FMU Interface for Modelica". In:

ControlledTemperature.fmu (2021). URL: https://github.com/modelica/fmi-cross-check/blob/master/fmus/2.0/cs/win64/Dymola/2017/ControlledTemperature/ControlledTemperature.fmu (visited on 2021-04-28).

DllExport (2021). URL: https://github.com/3F/DllExport (visited on 2021-04-23).

Drive Composer Pro (2021). URL: https://new.abb.com/drives/software-tools/drive-composer (visited on 2021-04-23).

DriveSize (2021). URL: https://new.abb.com/drives/software-tools/drivesize (visited on 2021-04-23).

Eclipse (2021). URL: https://www.eclipse.org/ide/ (visited on 2021-04-23).

Fabio, Cremona et al. (2017). "Hybrid co-simulation: it's about time". In: *Software and Systems Modeling* 18, pp. 1655–1679. DOI: https://doi.org/10.1007/s10270-017-0633-6.

Fei, Tao et al. (2018). "Digital twin-driven product design, manufacturing and service with big data". In: *The International Journal of Advanced Manufacturing Technology* 94, pp. 3563–3576. DOI: https://doi.org/10.1007/s00170-017-0233-1.

FMI standard (2021). URL: https://fmi-standard.org/ (visited on 2021-04-27).

FMI Toolbox import (2021). URL: https://www.modelon.com/products-services/modelon-deployment-suite/fmi-toolbox/ (visited on 2021-04-29).

fmi-standard tools (2021). URL: https://fmi-standard.org/tools/ (visited on 2021-04-27).

FMU compliance checker (2021). URL: https://github.com/modelica-tools/FMUComplianceChecker (visited on 2021-04-26).

Hauf, Dominik et al. (2017). "Multifunctional use of functional mock-up units for application in production engineering". In: *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pp. 1090–1095. DOI: 10.1109/INDIN.2017.8104925.

Hensel, Stephan et al. (2016). "Co-simulation with OPC UA". In: *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, pp. 20–25. DOI: 10.1109/INDIN.2016.7819127.

Jiapeng, Guo et al. (2019). "Modular based flexible digital twin for factory design". In: *Ambient Intelligence and Humanized Computing* 10, pp. 1189–1200. DOI: https://doi.org/10.1007/s12652-018-0953-6.

Lappalainen, Jari (2019). "Extending mechanistic thermal-hydraulic modellling and dynamic simulation for new industrial applications". Doctoral dissertation. Aalto University, Department of Chemical Engineering. DOI: http://urn.fi/URN:ISBN:978-952-60-8759-7.

Meer, A. A. van der et al. (2020). “Simulation-Based Assessment Methods”. In: *European Guide to Power System Testing: The ERIGrid Holistic Approach for Evaluating Complex Smart Grid Configurations*. Ed. by Thomas I. Strasser, Erik C. W. de Jong, and Maria Sosnina. Cham: Springer International Publishing, pp. 35–50. ISBN: 978-3-030-42274-5. DOI: 10.1007/978-3-030-42274-5_3. URL: https://doi.org/10.1007/978-3-030-42274-5_3.

Miettinen, T. (2012). “Synchronized Cooperative Simulation: OPC UA Based Approach”. Master’s thesis. Aalto University School of Electrical Engineering. URL: http://lib.tkk.fi/Dipl/2012/urn100571.pdf.

Motor-driven Equipment Research Package (2021). URL: https://omdia.tech.informa.com/-/media/tech/omdia/brochures/electric-motor-systems/fans-blowers-database---2020.aspx (visited on 2021-05-04).

Nishida, N., Y. A. Liu, and A. Ichikawa (1976). “Studies in chemical process design and synthesis II. Optimal synthesis of dynamic process systems with uncertainty”. In: *AIChE Journal* 22.3, pp. 539–549. DOI: https://doi.org/10.1002/aic.690220318.

Nouidui, Thierry, Michael Wetter, and Wangda Zuo (2014). “Functional mock-up unit for co-simulation import in EnergyPlus”. In: *Journal of Building Performance Simulation* 7.3, pp. 192–202. DOI: 10.1080/19401493.2013.808265. eprint: https://doi.org/10.1080/19401493.2013.808265. URL: https://doi.org/10.1080/19401493.2013.808265.

OPC UA Proxy (2021). URL: https://opcfoundation.org/products/view/opc-ua-proxy-1 (visited on 2021-04-23).

Sadjina, Severin et al. (2018-09). “Distributed Co-Simulation of Maritime Systems and Operations”. In: DOI: 10.1115/1.4040473.

SCL (2021). URL: https://dev.simantics.org/index.php?title=SCL_Tutorial (visited on 2021-04-23).

Simantics FMIL (2021). URL: https://gitlab.simantics.org/simantics/fmil (visited on 2021-04-23).

Simulink 2020b (2021). URL: https://se.mathworks.com/downloads/ (visited on 2021-04-28).

Simulink FMI import (2021). URL: https://se.mathworks.com/help/simulink/ug/work-with-fmi-in-simulink.html (visited on 2021-04-29).

Vega, P. et al. (2014). “Integrated design and control of chemical processes – Part I: Revision and classification”. In: *Computers and Chemical Engineering* 71, pp. 602–617. DOI: doi:10.1016/j.compchemeng.2014.05.010.

Virtual Drive (2021). URL: https://new.abb.com/drives/software-tools/drive-composer (visited on 2021-04-23).

Waide, Paul and Conrad Brunner (2011-01). “Energy-Efficiency Policy Opportunities for Electric Motor-Driven Systems”. In: URL: https://www.iea.org/reports/energy-efficiency-policy-opportunities-for-electric-motor-driven-systems.

Westerberg, Arthur (2004). “A retrospective on design and process synthesis”. In: *Computers and Chemical Engineering* 28, pp. 447–458. DOI: http://dx.doi.org/10.1016/j.compchemeng.2003.09.029.